MINDSET

Certified Developer Program

ISV Toolkit User's Guide

1 October 1984

MINDSET Corporation
617 N. Mary Avenue
Sunnyvale, CA 94086

(408)737-8555

Certified Developer Program

ISV Toolkit User's Guide

Version 1.0

1 October 1984

Information in this document is subject to change without notice and does not represent a commitment on the part of Mindset Corporation. It is against the law to copy the Mindset ISV Toolkit in part or in whole on magnetic tape, disk, or any other medium for any purpose other than the purchaser's licensed use.

MINDSET is a trademark of Mindset Corporation.

Printed in U.S.A.

Table of Contents

Table of Contents

Section 1

Introduction

General Information

Mindset has established the Certified Developer Program (CDP) to promote the development of software titles for the Mindset personal computer. Qualified Independent Software Vendors (ISV) participating in the CDP receive direct assistance from Mindset for software development Mindset has made available the following software devlopment documentation:

1. Introductory Guide to MS-DOS

2. Programmer's Development Library (PDL)

   Software Developer's Guide (SDG)
   MS-DOS Programmer's Reference Manual
   Macro Assembler Reference Manual

3. ISV Toolkit

   License for Toolkit Use
   Independent Software Vendor (ISV) Toolkit Guide
   Three diskettes containing the ISV Toolkit Libraries

The Introductory Guide to MS-DOS is a non-technical document intended to familiarize the reader with the Microsoft MS-DOS operating system. The Guide describes the function and use of most commands and utilities provided by MS-DOS. A more technical description of MS-DOS may be found in the PDL, MS-DOS Programmer's Reference Manual.

The PDL consists of documentation and program diskettes containing the most recent MS-DOS release and Mindset system utilities.

The SDG is a technical document describing the Mindset BIOS interrupt routines, and interface to the Mindset hardware. The ISV Toolkit program language libraries implement subroutine calls to these BIOS functions from high-level languages (C or PASCAL).

The MS-DOS Programmer's Reference Manual is a technical document describing the application program interface to the operating system.

The Macro Assembler Reference Manual is a technical document describing the assember program used on the Mindset computer system.

The Certified Developer Program kit contains the Mindset ISV Toolkit license and the ISV Toolkit.

The License defines the responsibilities of the ISV and limits the liability of Mindset.

The ISV Toolkit consists of the ISV Toolkit User's Guide, and three diskettes; containing the C and PASCAL language Toolkit libraries, the Interactive Drawing Aid (IDA) and a Sound Editor. The libraries each contain high-level language routines to access Mindset and MS-DOS interrupt functions. See Appendix A, Toolkit Diskette Configuration for a description of the contents of each of the three diskettes. Included on the C and PASCAL language library diskettes are all files required to develop C and PASCAL programs.

Example programs are supplied demonstrating the use of the ISV Toolkit and the capabilities of the Mindset computer system.

Purpose of User's Guide

Since the Toolkit is intended for use by experienced programmers, this Guide assumes a knowledge of, and makes frequent reference to, the Mindset documents listed previously. Additionally, the following language reference manuals may be required:

1) Microsoft/Lattice C Language Reference Manual

2) Microsoft PASCAL Language Reference Manual

It is assumed that the user has copies of the License, PDL and SDG, and appropriate language reference manual, as these documents are integral to the use of the Toolkit.

Two types of reference are provided by this Guide:

1) Detailed descriptions (by reference or example) of how to use MINDSET unique BIOS interrupt functions with C and PASCAL.

2) Appendix A, containing Useful Tables of frequently referenced information in easy to find form.

Overview of Library Use

The Toolkit libraries (C and PASCAL) allow development of programs in a high-level language while providing access to specific BIOS-level functions. Library routines are written in the high-level language (C and PASCAL) and are included in program modules in the same manner as any external routine. This is shown graphically in Figure 1-1.

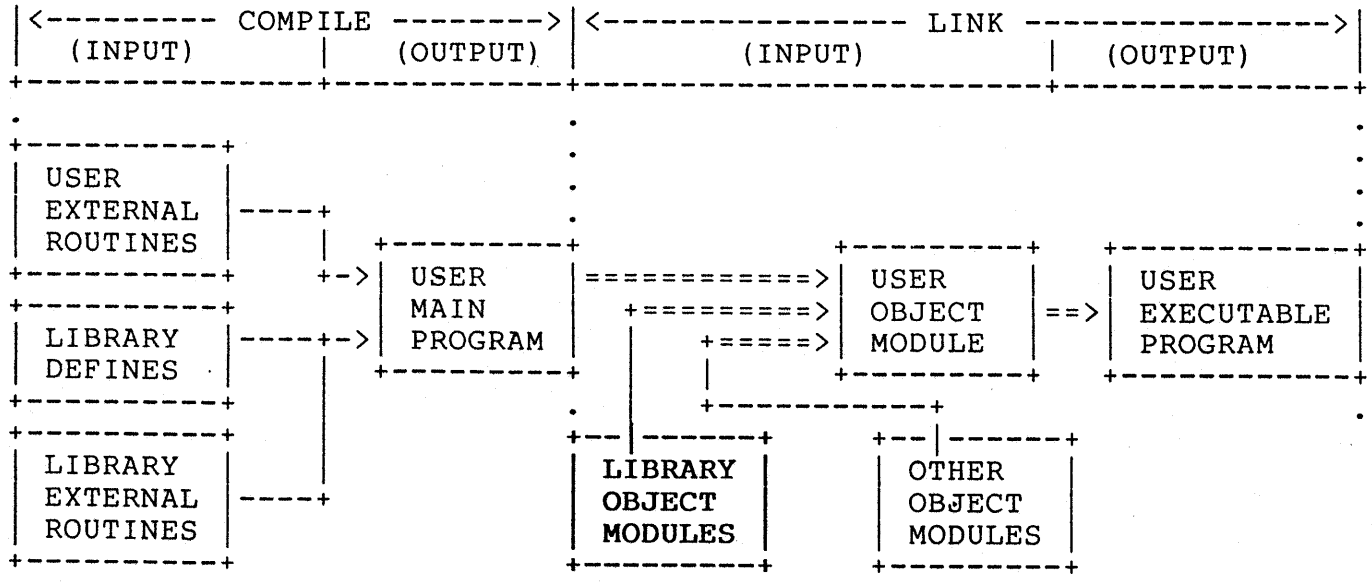


Figure 1-1 Context of Language Library

There are two primary steps for using the Toolkit library routines:

1) Compile the user program, including any external procedure references (to user or library routines) and the proper library define file. Refer to the examples in this guide, or to documentation provided with the C or PASCAL compiler in use.

2) Link the resultant user program object module with the supplied library object modules. This step will resolve all external procedure references included in the user main program. The output of the linker will be an executable user program. Refer to the examples in this guide, or to the Microsoft Linker documentation.

Guide to User's Guide

This ISV Toolkit Guide provides specific information on each library routine and includes appropriate references to other Mindset documents as required.

The first four sections in this Guide are ordered according to normal program development sequence;

Chapter 1 provides general information about the Certified Developer Program (CDP) and describes the format of this User's Guide.

Chapter 2 describes the reasons for using assembly language routines in a high-level language environment. Additional information on the Assembler is available in the Macro Assembler Manual in the PDL volume.

Chapter 3 presents the function of the Linker program and the significance of the Linker to the high-level language library. Additional documentation on the Linker is available in the Introductory Guide to MS-DOS in the PDL volume.

Chapter 4 describes the "how-to" of compiling high-level languages and linking the ISV Toolkit library to these programs. Additional information is available in the Microsoft/Lattice C Language Reference Manual, and the Microsoft PASCAL Language Reference Manuals.

The remaining chapters in this Guide are provided as references for high level language program development.

Chapter 5 contains documentation for each available high level language routine contained in the ISV Toolkit. Library routines are listed numerically within functional groups according to the corresponding interrupt function code associated with each routine. The Useful Tables in Appendix A contain an alphabetical list of all routines and a cross-reference of each routine to the corresponding BIOS function. The SDG in the PDL volumne describes each BIOS function in detail.

Chapter 6 contains Mindset Application Notes describing various system-level topics in detail. Tutorial in nature, each Application Note provides additional reference during program development. ISV requests for detailed information will determine topics addressed in future appplication notes.

Appendix A, Useful Tables, summarizes various groups of data in a concise table form. These tables are intended to provide quick reference during high level language program development. See the Table of Contents for a list of these tables.

Appendices B and C contain the source and batch execution files for several example programs. Appendix B contains the C language version of the example programs, and Appendix C the PASCAL language version. All programs funtions identically in both languages. See the Table of Contents for a list of example programs for each language.

Preparing the ISV Toolkit

Before you can begin program development, the ISV Toolkit must be prepared for use. Preparation consists of two major activities:

1) make copies of the ISV Toolkit Library diskettes,

2) create the Program Development Environment diskette.

A copy of each of the two ISV Toolkit Language Library diskettes should be made from the originals in the ISV Toolkit. These copies will be used for program development. The original diskettes should be stored, and used only to create new copies as required. The following steps demonstrate the sequence for copying these diskettes.

1) Boot the Mindset computer system, using the MS-DOS system diskette supplied with the PDL.

2) Format a double-sided, double-density diskette using the Mindset FORMAT utility program on the MS-DOS system diskette.

3) Place this formatted diskette in drive B: (right-hand drive).

4) Place the C Toolkit Library diskette in drive A: (left-hand drive).

5) Select drive A: as the default drive by entering:

```
n>A: <return>
A>
```

where **n>** is the MS-DOS prompt, and may appear as A> or B>, and **<return>** means the key labelled **return** is to be pressed.

6) Copy the contents of the C Toolkit Library diskette in drive A: to the formatted diskette in drive B: with the COPY command:

```
A>COPY A:*.* B: <return>
A>
```

7) Repeat steps 2 - 6 to make a copy of the PASCAL Toolkit Library diskette.

This completes the process for creating copies of the ISV Toolkit Library diskettes. Use these copies during program development. Store the original diskettes, using them only to make new copies for program development.

Creating the Program Development with the ISV Toolkit requires two diskettes:

1) copy of appropriate ISV Toolkit Language Library diskette,

2) a Program Development Environment diskette.

The Program Development Environment diskette contains the compiler for the language you are using (C or PASCAL), the Microsoft Linker, and a text editor program for creating and editing programs. This diskette may be created similarly to the Toolkit Library diskettes; FORMAT a double-sided, double-density diskette with the Mindset FORMAT utility, then copy the recommended files from your diskettes to the Program Development Environment diskette.

Refer to the Program Development Environment table in Appendix A for the recommended list of files that should be placed on this diskette.

You do not have to create the Program Development Environment diskette, although doing so will reduce the number of times you need to exchange diskettes in drive A: while developing a program (as you switch between the editor and compiler).

If you choose NOT to make this diskette, you may use the compile link and go batch files (CC.BAT and PASCAL.BAT), ONLY if the compiler and linker are on the same diskette.

Program Development Sequence

This Guide assumes that you have created a Program Development Environment diskette. Throughout program development, this diskette will remain in drive A: (left-hand drive), while Toolkit Library copy diskette will remain in drive B: (right-hand drive). The following steps demonstrate how to use a batch file to compile, link and run one of the example programs.

1) Type in the compile, link and go batch file for the example program POINT. Listings of the DOPOINT.BAT, POINT.C (and POINT.PAS) are included in this Guide (see Appendix B and C). POINT.C (and POINT.PAS) are included with the ISV Toolkit.

2) Test the Program Development Environment by executing the batch file DOPOINT.BAT and verifying correct compilation, link and output for the POINT program (a single white dot is drawn on the left third of the display screen, about half-way down). Pressing any key on the keyboard will exit the program.

3) After successfully completing step 2, copy the source file for POINT to a new source file named CIRCLE (eg. CIRCLE.C or CIRCLE.PAS). Modify CIRCLE to display a hollow, white circle (refer to Useful Tables, Shapes to find the proper Mindset library function for drawing a hollow circle). Hint, replace the call to **blt_polypoint** with a call to **blt_hellipses** making sure that you establish the correct arguments.

4) Type in the generic compile, link and go batch file (eg. CC.BAT or PASCAL.BAT), as shown in this User's Guide, Examples. Test this batch file by using it to execute CIRCLE eg. B> CC CIRCLE).

5) If steps 1 - 4 have been accomplished without frustration, then proceed with the development of MINDSET software. If any problems are encountered, then please call MINDSET immediately at (408)737-8555, and mention that you are in the Certified Developer Program. The CDP is intended to provide timely support for ISV program development efforts.

6) Review the printed copies of all example programs (Appendix B or C), for ideas in the use of more complex library routines.

Current Library

The Numeric and Alphabetic lists in Appendix A provide a complete list of MINDSET, IBM and MS-DOS interrupts and the corresponding name of the Toolkit Library routine. Notice that not all of the listed interrupt functions have accompanying library routines. There are two possible reasons:

1) The interrupt function is provided in the high level language (eg. file operations).

2) The current release of the ISV Toolkit does not contain a library routine for this interrupt. This situation may occur when ISV interest in this interrupt is insufficient to justify implementing a library routine. Part of the CDP is to elicit response from ISV's with respect to implementing additional interrupt library routines and higher level functional library routines.

As a further note, there are 4 versions of the C library object modules. Refer to the Microsoft (or Lattice) C reference documentation on these program and data memory size models to determine which model is correct for a particular program. The current ISV Toolkit C language library supports only the small memory model.

User Feedback

An ISV participating in the CDP has agreed in the License to report any defects discovered in MINDSET supplied utility software. ISV's are also encouraged to inform Mindset of the following:

1) Requirement(s) for high-level language routines with higher functionality than current library routines. It is the intent of MINDSET to develop and provide such routines.

2) Incomplete or misleading documentation. Mindset wishes to provide accurate support to maximize ISV productivity.

3) Any technical questions or possible topics for application notes. Mindset wishes to provide the information necessary for successful completion of ISV products.

Future Library Development

Future additions to the Toolkit library are planned, including implementation of more BIOS interrupt functions, and higher-level functions. Currently, graphics is assigned the highest priority, with communications and mouse applications following.

These priorities are subject to change depending mainly on ISV feedback.

Section 2

Assembly Language Routines

Reasons for Using Assembly Language Routines

There are two major reasons for using assembly language routines with high-level language interface routines:

1) minimize code segment size, for speed (see item 2) or due to memory constraints for a particular application,

2) shorten execution time, for overall code performance or due to existing constraints (eg. collision detect, vertical retrace time, and display interrupt routines are constrained by fixed time periods in which to execute).

The time constraints in item 2 are particularly important for animation routines and user-controlled cursor applications (eg. a mouse pointer). Routines performing animation or using the entire display screen must synchronize with the display interrupt (refer to Useful Tables, Toolkit Cross Reference) to avoid flickering of the display. The display interrupt is generated every 1/60 second by the Mindset video display processor.

Refer to the SDG, Section 4, in the section on VBLANK Operations for additional technical details and timing information relating to the display interrupt.

Use of Assembly Language Routines

Assembly language routines included in the ISV Toolkit are provided in source and object form. Each source routine has been assembled with the Microsoft Assembler (MASM) to obtain the corresponding object module. Any high-level language program requiring an assembly language routine must have the routine defined as external. The program is then compiled in the normal manner (see the Examples or Section 4). After succesful compilation, the program is linked with the assembly routine to complete the executable program image (see Appendices B and C, or Section 3 in this Guide).

Additional information on the assembly language used by the Mindset computer system may be found in the Macro Assembler section of the PDL volume.

Assembly Language Routine Library

The following assembly language routines are provided with the ISV Toolkit:

1) callbios - C language interface to BIOS

2) bios_interrupt - PASCAL language interface to BIOS

3) evbinit - turns on the system interrupt generated at each vertical retrace

4) stopevb - turns off the vertical retrace interrupt

5) bltchar - performs a bit block transfer from a font image to the display buffer

The format used to document each of these routines is described more fully in Section 4, Language Library.

Assembly Language Routines

NAME

callbios -- C language library interface routine to Mindset BIOS

SYNOPSIS

C Language

```
int     mindset_interrupt;      Mindset function interrupt
                                (either EE or EF)
int     *mindset_registers;     Pointer to structure defining
                                Mindset register structure.
                                This structure is myregs, and
                                is defined in <cuser.inc>
callbios(mindset_interrupt,&mindset_registers);
```

REFERENCE

See the MINDSET SDG for BIOS functions.

CAUTIONS

See the References for possible error conditions not returned.

REVISION

Version	Date	Comments
Original	07/01/84	Original version

Assembly Language Routines

NAME

bios_interrupt - PASCAL language interface routine to Mindset BIOS

SYNOPSIS

PASCAL Language

```
var     interrupt: integer;     Interrupt number to be used.
                                (Mindset unique are EE and EF).
var     reg: reg_block;         Pointer to structure defining
                                Mindset registers.

bios_interrupt(interrupt,reg);
```

REFERENCE

See the MINDSET SDG document for BIOS functions.

CAUTIONS

See the References for possible error conditions not returned.

REVISION

Version	Date	Comments
Original	07/01/84	Original version

Assembly Language Routines

NAME

evbinit -- Enables interrupt generated by video vertical retrace.

SYNOPSIS

C Language	PASCAL Language
evbinit( );	evbinit( );

REFERENCE

See the MINDSET SDG Section 4, Display Interrupt Control and VBLANK operations.

CAUTIONS

See the References for possible error conditions not returned.

REVISION

Version	Date	Comments
Original	07/01/84	Original version

Assembly Language Routines

NAME

stopevb -- disables generation of interrupt by vertical retrace.

SYNOPSIS

C Language	PASCAL Language
stopevb( );	stopevb( );

REFERENCE

See the MINDSET SDG Section 4, Display Interrupt Control and VBLANK Operations

CAUTIONS

See the References for possible error conditions not returned.

REVISION

Version	Date	Comments
Original	07/01/84	Original version

Assembly Language Routines

NAME

bltchar -- performs bit block move from character font to display memory.

SYNOPSIS

C Language

```
char    *text;          pointer to text to display
int     size;           length of text
int     color;          palette index of color to display text
int     xorigin;        pixel location of upper-left-hand point of
                        first character in text on x-axis
int     yorigin;        pixel location of first character on y-axis
int     fontdesc;       offset of memory containing font description

x = bltchar(*text, size, color, xorigin, yorigin, fontdesc);
                        x-axis location of next character block is
                        returned
```

REFERENCE

See the MINDSET SDG Section 4, Custom Character Set Operations.

CAUTIONS

See the References for possible error conditions not returned.

REVISION

Version	Date	Comments
Original	07/01/84	Original version

Section 3

LINKER/COMPILERS

Function of Linker

The function of the Linker is to resolve external routine and variable references in a program. A routine or variable is said to be external to a program if the routine or variable is not defined in the program. Program development is made easier by developing routines individually, then using these routines to develop a program.

Such a structured approach is the basis for the ISV Toolkit. Each routine has been separately developed, then assembled into an object module suitable for linking. High-level language user programs may call these routines to perform specific functions by defining the routines as external to the main program and linking them with the main program after successful compilation of the user program.

Overview of Linker Use

To simplify the use of Linker with the ISV Toolkit, all routines have been assembled into a single place, a link library. Each library has a filename extension of .LIB (eg. BIOS.LIB is the name of the PASCAL and C link library). The link library simplifies the use of the Linker by reducing the number of arguments given to the Linker (refer to the example sections in this chapter).

When a link library name is given to the Linker, the Linker searches for all external routine references. Those external routines which are found by this search are linked to the main program. If an external routine is not found in the link library, then a separate object module must exist for that routine and the name of the object module must be included with the main program name as an argument to the Linker.

Any external routine not found in one of these two places will cause an 'Unresolved External' error message to be displayed by the Linker.

As shown in Appendix B and C, using a link library allows for one simple batch file for compiling and linking of all user programs.

```
********************* I M P O R T A N T   N O T E *********************
*
*  Even though the link library name (BIOS.OBJ) is the same for both *
*  C and PASCAL, the two link libraries are NOT interchangeable.      *
*
*  Refer to Appendix A, ISV Toolkit Diskette Configuration and        *
*  Program Development Environment, for additional information.       *
*
***************************************************************************
```

Example with C Library

The example program POINT.C may be compiled and linked with the following batch file. Note that each external routine is named individually with the main program in the first Linker argument list. POINT.C is listed in Appendix B, and is contained on Diskette 1 of 3.

BATCH FILE LINE	COMMENT
lc1 point -n	Invoke Lattice C compiler, Pass #1. Compile the C program POINT.C (compiler assumes .C filename extension). The -n allows variable names longer than 8 characters (maximum of 32).
lc2 point	Invoke Lattice C compiler, Pass #2. The object module POINT.OBJ is output from this pass.
link cs point points getkey setmode setibmmode,point,,lcs;	The first Linker argument is a list of object modules to link. CS is the Lattice C small model; code and data occupy less than 64Kbytes. Point is the main program object module from the second pass of the compiler. Points, getkey, setmode, setibmmode are all separately compiled (if written in C) object modules which are called by POINT. The second Linker argument (after the comma, is the executable code name (POINT.EXE) in this case. The third Linker argument is blank, since no map file is desired. LCS is the Lattice small model link library.

Another approach is to use a generic batch file (CC.BAT in the ISV Toolkit), and the MINDSET link library (BIOS.OBJ). The generic batch file is intended to compile, link and run a single C language source file.

The generic batch file (CC.BAT) is as follows:

```
B>TYPE CC.BAT

lc1 %1 -n
lc2 %1
link cs %1,%1,,lcs bios
%1

B>
```

Refer to Appendix A, Program Development Environment for additional details on what the contents of diskettes in drive A: and B: should be. See the Introductory Guide to MS-DOS for more details on batch files. To effect the compilation of POINT.C, enter:

B> CC POINT

This command invokes CC.BAT to compile, link and run POINT. This has the same effect as the previous example. Note the major difference is in the last Linker argument; LCS is accompanied by BIOS (C language link library). All routines called by POINT are included in BIOS (the full filename is BIOS.OBJ).

Similarly, any single C language program may be compiled, linked and run, by entering;

B> CC <filename>

where **<filename>** is the name of a C program. Filename may be entered as <filename>.C, but this is not required.

See the Microsoft/Lattice C compiler reference manual for detail description of the 4 memory models which are supported.

See Appendix B in this document for additional program examples using the CC.BAT batch file.

Example with PASCAL Library

The example program POINT.PAS may be compiled and linked with the following batch file. Note that each external routine is named individually with the main program in the first Linker argument list. Appendix C lists POINT.PAS, and Diskette 2 of 3 contains the source.

BATCH FILE LINE	COMMENT
pas1 point	Invoke PASCAL compiler, Pass #1. Compile the PASCAL program POINT.PAS (compiler assumes .PAS filename extension).
pas2	Invoke PASCAL compiler, Pass #2. The object module POINT.OBJ is output from this pass. Pass 2 does not require any arguments, since PASIBF.BIN and PASIBF.SYM are assumed to exist from Pass 1 of the compiler. These files are used by Pass 2.
link point points getkey setmode setibmmode,point,,pascal;	The first Linker argument is a list of object modules to link. Point is the main program object module from the second pass of the compiler. Points, getkey, setmode, setibmmode are all separately compiled PASCAL program modules which are called by POINT. The second Linker argument (after the comma, is the executable code name (POINT.EXE) in this case. The third Linker argument is blank, since no map file is desired. Pascal is the PASCAL language runtime link library.

Alternatively, you may use a generic batch file (PASCAL.BAT in the ISV Toolkit), and the MINDSET link library (BIOS.OBJ). The generic file is intended to compile, link and run a single PASCAL language source program file.

The generic batch file (PASCAL.BAT) is as follows:

```
B>TYPE PASCAL.BAT

pas1 %1
pas2
link %1,%1,,pascal bios
%1

B>
```

Refer to Apendix A, Recommended Development Environment for more details on what the contents of diskettes in drive A: and B: should be. Additional information about batch files and parameters may be obtained in the Introductory Guide to MS-DOS in the PDL volume.
To effect the compilation of POINT.PAS, enter:

B> PASCAL POINT

This command invokes PASCAL.BAT to compile, link and run POINT, with the same effect as the previous example. Note the major difference is in the last Linker argument; PASCAL is now accompanied by BIOS (the MINDSET link library). All routines called by POINT are included in BIOS (the full filename is BIOS.OBJ).

Similarly, any single PASCAL language program may be compiled, linked and run by entering,

B> PASCAL <filename>

where **<filename>** is the name of a PASCAL language program, and may be entered as <filename>.PAS, but this is not required.

Refer to the Microsoft PASCAL compiler reference manual for additional detail.

Refer to Appendix C in this document for additional program examples using the PASCAL.BAT batch file.

Section 4

Language Library

Language Library

The ISV Toolkit contains two language libraries;

1) C language library
2) PASCAL language library

for use in developing application programs for the MINDSET computer. Both libraries have identical contents, one compiled in C, the other compiled in PASCAL.

Both libraries have the same name, BIOS.OBJ (refer to Section 3 in this document for a discussion on the use of BIOS.OBJ and the Linker program).

Corresponding routines in each library have been given the same name for traceability and field support purposes. Therefore, the same results should be obtained by calling the routine from either C or PASCAL. If any deviations are noted, Mindset should be informed immediately, as stipulated in the License (see Section 1 in this document).

A brief explanation of procedures for compiling, linking and running source programs is provided in Section 3 of this document.

Examples of compiling, linking and running high-level language source programs in C and PASCAL may be found in Appendices B and C of this document.

Library Format

The following format is used to describe each routine in the BIOS link library. The format is designed to provide a functional and narrative description. The BIOS link library routines are listed in numerical order. Appendix A contains an index of all routines in alphabetical order and cross reference to BIOS functions.

NAME

INTERRUPT #= nn
VALUE OF AH= nn

Routine name -- brief description (1 line)

SYNOPSIS

C Language

```
extern   arg1;
int      arg2,arg3;
    .
    .
    .
    .
routine(arg1,arg2...argn)
```

PASCAL

```
type   arg1;
var    arg2, arg3;
    .
    .
    .
    .
routine(arg1,arg2...argn);
```

REFERENCE

References to the MINDSET SDG or MS-DOS documents for basic function.

DESCRIPTION

Long description of routine, including typical usage and references to include files, model being used, etc.

CAUTIONS

Specific precautions applicable to routine.

REVISION

Revision history of routine.

Version	Date	Comments
Original	dd/mm/yy	Original version
V0.1	DD/MM/YY	Corrected previous bug

Library

The remainder of this chapter contains descriptions (see the previous section) of all library routines. Four subsections are provided, each containing those library routine descriptions corresponding to the functional area covered in the subsection.

In this chapter, the routines are ordered numerically according to the following functional groups:

1) Mindset Unique Routines (INT EE)
2) Mindset Graphics Routines (INT EF)
3) IBM Compatible Routines
4) MS-DOS Function Routines (INT 21)

For cross reference purposes, Appendix A contains a complete listing ordered by interrupt number, sub-ordered by function value in register AH (see to Appendix A, Numeric List of Library Routines). Appendix A also contains a complete listing of all routines in alphabetical order.

```
********************* I M P O R T A N T   N O T E **********************
*                                                                        *
*   All C language programs must include the MINDSET library structures  *
*   definitions.  This file is named CUSER.INC, on Diskette 1 of 3.      *
*                                                                        *
*   For example:                                                         *
*                                                                        *
*        /* This is the first line in the C language program file */     *
*        #include <cuser.inc>          /* include structures      */     *
*                                                                        *
*        /* The remainder of the program follows                 */     *
*              .                                                         *
*              .                                                         *
*              .                                                         *
*         main()                                                         *
*         {    .                                                         *
*              .                                                         *
*         }    .                                                         *
*         /* This is the last line in the C language program file */     *
*                                                                        *
**************************************************************************
```

```
********************* I M P O R T A N T   N O T E **********************
*                                                                        *
*   All PASCAL programs must include the MINDSET library structures      *
*   definitions.  This file is BIOS.TYP on Diskette 2 of 3.              *
*                                                                        *
*   For example:                                                         *
*                                                                        *
*        { PROG.PAS  - Any PASCAL program.}                              *
*                                                                        *
*        program prog(input,output);                                     *
*                                                                        *
*         const                                                          *
*          CONST = 1;       { Set up constants}                        *
*                                                                        *
*         type                                                           *
*          TYPE0 =          { Set up types}                            *
*            .                                                           *
*            .                                                           *
*            .                                                           *
*          {$include:'bios.typ'}                                         *
*                                                                        *
*         var                                                            *
*          var0 : byte;     { Set up variables.}                       *
*            .                                                           *
*            .                                                           *
*        (Remainder of program)                                          *
*                                                                        *
**************************************************************************
```

NAME

Interrupt # = 10
Value of (AH)= 00

set_ibm_mode - Sets video display mode via industry standard function.

SYNOPSIS

--- C ---

char video_mode; IBM-compatible video mode

--- PASCAL ---

video_mode: byte;

0 - 40 x 25 Black/White
1 - 40 x 25 Color
2 - 80 x 25 Black/White
3 - 80 x 25 Color
4 -320 x 200 Color
5 -320 x 200 Black/White
6 -640 x 200 Black/White
7 - 80 x 25 Black/White (Monochrome adapter)

set_ibm_mode(video_mode);

REFERENCE

MINDSET SDG Section 3 (Video I/O -- Interrupt 10H).

DESCRIPTION

This routine selects the desired IBM-compatible video display mode.

CAUTIONS

Whenever the video display mode is changed, the display buffer is cleared by filling it with spaces (character modes), or with zeroes (graphics modes).

The display page is set to page 0.

The cursor is initialized to lines 6 and 7 (in an 8 x 8 pixel cell).

Register AL in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 10
Value of (AH)= 02

set_cursor_position - Sets new cursor page, row and column position. Cursor displays at new position.

SYNOPSIS

--- C ---		--- PASCAL ---
char page;	Set display page cursor on	**page: byte;**
char row;	Select row cursor is on	**row: byte;**
char column;	Select column cursor is in	**column: byte;**

```
set_cursor_pos(page, row, column);
```

REFERENCE

MINDSET SDG Section 3 (Video I/O -- Interrupt 10H).

DESCRIPTION

This routine specifies the page, column and row for the cursor. The cursor appears at the specified (column,row) when the specified page is displayed.

A (column,row) specification of (0,0) indicates the upper left corner of display screen.

CAUTIONS

In all modes, the display processor masks the specified page to a legal value for the current display mode (See SDG Section 3, Character Mode Operation and Graphics Mode Operation).

The value for row is automatically limited to the maximum for the current display mode (See SDG Section 3, Video I/O -- Interrupt 10H, Set Mode). The minimum row value is always 0.

The value for column is automatically limited to the maximum for the current display mode (See SDG Section 3, Video I/O - Interrupt 10H, Set Mode). The minimum column value is always 0.

See also, get_cursor_position (Interrupt # = 10, Value of (AH) = 03).

Registers AL,BH,DX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 10
Value of (AH)= 03

get_cursor_position - Get current cursor page, row and column position.

SYNOPSIS

--- C ---		--- PASCAL ---
char page;	Set display page for cursor	page: byte;
char *row;	Return row cursor is on	var row: byte;
char *column;	Return column cursor is in	var column: byte;

```
get_cursor_position(page, row, column);
```

REFERENCE

MINDSET SDG Section 3 (Video I/O -- Interrupt 10H).

DESCRIPTION

This routine returns the current column and row for the cursor on the page specified.

A (column,row) specification of (0,0) indicates the upper left corner of display screen.

See also, set_cursor_position (Interrupt # = 10, Value of (AH) = 02).

CAUTIONS

In all modes, the display processor masks the specified page to a legal value for the current display mode (See SDG Section 3, Character Mode Operation and Graphics Mode Operation).

Registers AL,BH,CX,DX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 10
Value of (AH)= 09

write_char_only - Write a character with attributes to the current cursor position.

SYNOPSIS

--- C ---		--- PASCAL ---
char page;	Set display page for cursor	**page: byte;**
char chr;	ASCII character to display	**chr: byte;**
char color;	Character color palette index	**color: byte;**
char count;	Number of times character is written on the row	**count: byte;**

write_char_only(page, chr, color, count);

REFERENCE

MINDSET SDG Section 3 (Video I/O -- Interrupt 10H).

DESCRIPTION

Write_char_only displays the ASCII character at the current cursor position (See SDG Section 3, Video I/O - Interrupt 10,SET CURSOR POSITION).

If the value of count is greater than 1, then the character is duplicated count times. All characters must remain on the same row.

If the value of color is 128 or greater (eg. 128 + color palette index), then the color palette index specified is exclusive-ORed (XOR) with the current color at the current cursor position. This applies only in graphics modes.

CAUTIONS

In all modes, the display processor masks the specified page to a legal value for the current display mode (See SDG Section 3, Character Mode Operation and Graphics Mode Operation).

Registers AL,BH,CX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 10
Value of (AH) = 0A

write_char_cursor - Writes character at the current cursor position.

SYNOPSIS

--- C ---		--- PASCAL ---
char page;	Set display page for cursor	page: byte;
char chr;	ASCII character to display	chr: byte;
char color;	Character color palette index	color: byte;
int count;	Number of characters to be written on row	count; integer;

```
write_char_cursor(page, chr, color, count);
```

REFERENCE

MINDSET SDG Section 3 (Video I/O -- Interrupt 10H).

DESCRIPTION

Write_char_cursor displays the ASCII character at the current cursor position (See SDG Section 3, Video I/O - Interrupt 10,SET CURSOR POSITION).

If the value of count is greater than 1, then the character is duplicated count times. All characters must remain on the same row.

If the value of color is 128 or greater (eg. 128 + color palette index), then the color palette index specified is exclusive-ORed (XOR) with the current color at the current cursor position. This applies only in graphics modes.

CAUTIONS

In all modes, the display processor masks the specified page to a legal value for the current display mode (See SDG Section 3, Character Mode Operation and Graphics Mode Operation).

Registers AL,BH,CX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 10
Value of (AH)= 0C

write_dot - Writes a dot (single pixel) with specified color at the the specified row and column.

SYNOPSIS

--- C ---		--- PASCAL ---
int row;	Set pixel row to display dot	**row: integer;**
int column;	Set pixel column to display dot	**column: integer;**
char color;	Dot color palette index	**color: byte;**

write_dot(row, column, color);

REFERENCE

MINDSET SDG Section 3 (Video I/O -- Interrupt 10H).

DESCRIPTION

Write_dot displays a single pixel at the specified pixel row and column position. See SDG Section 3, Graphics Mode Operation, for details on graphics modes and color palette use.

If the value of color is 128 or greater (eg. 128 + color palette index), then the color palette index specified is exclusive-ORed (XOR) with the current color at the current cursor position. This applies only in graphics modes.

See also, read_dot (Interrupt # = 10, Value of (AH) = 0D).

CAUTIONS

Registers AL,CX,DX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 10
Value of (AH)= 0D

read_dot - Returns the color of the dot (single pixel) at the specified pixel row and column position.

SYNOPSIS

--- C ---		--- PASCAL ---
int row;	Set pixel row to display dot	row: integer;
int column;	Set pixel column to display dot	column: integer;
char color;	Color of dot returned in PASCAL	var color: byte;
color = read_dot(row,column);		read_dot(row,column,color)

REFERENCE

MINDSET SDG Section 3 (Video I/O -- Interrupt 10H).

DESCRIPTION

Read_dot routine returns the color palette index of the pixel specified by the pixel row and column position.

See also, write_dot (Interrupt # = 10, Value of (AH) = 0C).

CAUTIONS

This routine has no effect in character modes (See SDG Section 3, Character Mode Operation).

Registers AL,CX,DX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 10
Value of (AH) = 0E

write_teletype - Writes an ASCII character at the current cursor position with the specified color palette index. The cursor position is updated by this routine.

SYNOPSIS

--- C ---		--- PASCAL ---
char page;	Set page for character display	**page: byte;**
char chr;	ASCII character to display	**chr: byte;**
char color;	Character color palette index	**color: byte;**

write_teletype(page, chr, color);

REFERENCE

MINDSET SDG Section 3 (Video I/O -- Interrupt 10H).

DESCRIPTION

Write_teletype displays the ASCII character at the current cursor position. The cursor position is updated as required before, and after the character is displayed.

Normally, the updated cursor position is the next column on the current row of the display.

If the cursor is at the maximum column position for the current display mode, then the cursor is advanced to the next row, in column 0, where the ASCII character is then displayed.

If the ASCII character is a line feed, and the cursor is at the maximum row position for the current display mode, then the entire display is scrolled up one line. The new line is filled with blanks (in character modes) or zeroes (in graphics modes).

This routine implements 4 control characters (carriage return, line feed, bell, and backspace).

See SDG Section 3, Video I/O -- Interrupt 10H, WRITE TELETYPE, for more detail on this routine.

CAUTIONS

Contrary to industry standard, this routine will work for any valid page on the MINDSET.

Registers AL,CX,DX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 16
Value of (AH)= 00

get_kb_char - Read the next keystroke from the keyboard.

SYNOPSIS

--- C ---		--- PASCAL ---
char chr;	Character from keystroke	var chr: char;
char *scancode;	Scan code of keystroke	var scan_code: byte;
chr = get_kb_char(scancode);		get_kb_char(chr, scan_code)

REFERENCE

MINDSET SDG Section 3 (Video I/O -- Interrupt 16H).

DESCRIPTION

Get_kb_char routine returns the keyboard scancode and ASCII value of the first keystroke after the routine is called. See Useful Tables, Keyboard Scan Codes for the corresponding keystrokes, scan codes and ASCII values.

CAUTIONS

Register AX in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 16
Value of (AH) = 01

test_kb_buffer - Indicates if an ASCII character is available to be read from the keyboard.

SYNOPSIS

--- C ---		--- PASCAL ---
char chr_present;	Zero if buffer empty	chr_present: boolean;
	ASCII character	chr: char;
	Scan code of character	scan_code: byte;
chr_present = test_kb_buffer();		test_kb_buffer(chr_present, chr,scan_code)

REFERENCE

MINDSET SDG Section 3 (Video I/O -- Interrupt 16H).

DESCRIPTION

Test_kb_buffer routine test the keyboard I/O processor for an available scan code. If scan_code is available, then a value of 1 is returned, otherwise a value of 0 (no scan_code available). A scan_code is available when a key has been pressed on the keyboard, but has not been read by the 80186 CPU.

See SDG Section 3, Keyboard I/O - Interrupt 16H.

CAUTIONS

Register AX in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 16
Value of (AH)= 02

get_kb_shift_status - Returns the current status of several keys on the keyboard.

SYNOPSIS

--- C ---

```
char kb_flag;  Bit(s) set for certain keys

kb_flag  = get_kb_shift_status();
```

--- PASCAL ---

```
var kb_flag: byte;

get_kb_shift_status(kb_flag);
```

REFERENCE

MINDSET SDG Section 3, Keyboard I/O --- Interrupt 16H.

DESCRIPTION

Get_kb_shift_status returns a bitmap describing the current status of several keys on the keyboard.

Bit(s)	Bit Name	Meaning (if Bit = 1)
0,1,2	N/A	(unused)
3	Hold State	PAUSE key has been toggled.
4	Scroll State	SCROLL LOCK key is depressed.
5	Num Shift	NUM LOCK key is depressed.
6	Caps Shift	CAPS LOCK key is depressed.
7	Ins Shift	INSERT KEY is depressed.

In the above table, the bit position is set to '1' if the key is currently depressed. The PAUSE key bit is set whenever the PAUSE key is pressed, since this key does not latch in the depressed position.

CAUTIONS

Register AX in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 21
Value of (AH)= 3C

dos_file_create - Create a new MS-DOS file, by entering the name in the current directory and setting the file length to 0.

SYNOPSIS

```
--- C ---                                          --- PASCAL ---

int offset;   Offset of pointer to path name       offset: word;
              (drive, directory, file name)
int segment;  Segment of pointer to path name      segment: word;
int attrib;   Bits indicate file attributes        attrib: integer;
int result;   Value indicating MS-DOS result       var result: integer;
                 0 - failure
                -3 - path not found
                -4 - too many open files
                -5 - access denied
                >0 - file handle of file

result = dos_file_create(offset,                   dos_file_create(offset,
                 segment, attrib);                   segment, attrib, result);
```

REFERENCE

MINDSET MS-DOS Programmers Reference Manual, Section 1, Create a File.

DESCRIPTION

Dos_file_create issues a request to MS-DOS to make a new directory entry with the specified pathname and attributes.

If the specified pathname is already in the current directory, then the file length for that pathname is set to 0.

The routine indicates failure by returning a 0.

If dos_file_create is successful then the handle (from MS-DOS) is returned with the file opened for read/write access.

CAUTIONS

Registers AX,CX,DX,DS in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 21
Value of (AH) = 3D

dos_file_open - Open the specified file with the requested access rights. The MS-DOS file handle is returned for the file.

SYNOPSIS

--- C ---

```
int offset;   Offset of pointer to path name
int segment;  Segment of pointer to path name
int access;   Code for file access rights

              Code      Access
              ----      ------
                0        Read file
                1        Write file
                2        Read and write file

 int result;  Result of MS-DOS operations
                0 - failure
               -2 - file not found
               -4 - too many open files
               -5 - access denied
              -12 - invalid access
               >0 - file handle of file

 result = dos_file_open(offset,
              segment, access);
```

--- PASCAL ---

```
offset: word;
segment: word;
access: integer;

var result: integer;

dos_file_open(offset,
   segment, access, result);
```

REFERENCE

MINDSET MS-DOS Programmers Reference Manual, Section 1, Open a File.

DESCRIPTION

Dos_file_open issues a request to MS-DOS to open the pathname with the requested access rights.

If the specified pathname is already in the current directory, then the file handle is returned.

The routine indicates failure by returning a 0.

CAUTIONS

Registers AX,CX,DX,DS in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 21
Value of (AH) = 3E

dos_file_close - Close the file specified by the MS-DOS file handle.

SYNOPSIS

--- C ---		--- PASCAL ---
int handle;	file handle to be closed	**handle: integer;**
int result;	Result of MS-DOS operations 0 - failure -6 - invalid handle	**var result: integer;**
result = dos_file_close(handle);		**dos_file_close(handle, result);**

REFERENCE

MINDSET MS-DOS Programmers Reference Manual, Section 1, Close a File.

DESCRIPTION

Dos_file_close issues a request to MS-DOS to close the file with the specified file handle.

All MS-DOS buffers for the file are flushed.

CAUTIONS

Register AX in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 21
Value of (AH) = 3F

dos_file_read - Read the specified number of bytes from the given file handle.

SYNOPSIS

--- C ---

```
int handle;File handle to read from
int offset;Pointer to destination buffer
int segment;Segment of destination buffer
int nbytes;  Number of bytes to read
int result;  Result of MS-DOS operations
               0 - tried to read end of file
              -5 - access denied
              -6 - invalid handle

result = dos_file_read(handle, offset,
           segment, nbytes);
```

--- PASCAL ---

```
handle: integer;
offset: integer;
segment: integer;
nbytes: integer;
var result: integer;

dos_file_read(handle, offset,
    segment, nbytes, result);
```

REFERENCE

MINDSET MS-DOS Programmers Reference Manual, Section 1, Read from File.

DESCRIPTION

Dos_file_read issues a request to MS-DOS to read the specified number of bytes from the given file handle. The bytes read are stored in the buffer pointed to by segment and offset.

If possible, the number of bytes specified will be read.

CAUTIONS

Registers AX,BX,CX,DX,DS in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 21
Value of (AH) = 40

dos_file_write - Write the specified number of bytes to the given file handle.

SYNOPSIS

--- C ---

```
int handle;   File handle to write
int offset;   Pointer to source buffer
int segment;  Segment of source buffer
int nbytes;   Number of bytes to write
int result;   Result of MS-DOS operations
                0 - failure
               -5 - access denied
               -6 - invalid handle

result = dos_file_write(handle, offset,
             segment, nbytes);
```

--- PASCAL ---

```
handle: integer;
offset: word;
segment: word;
nbytes: integer;
var result: integer;

dos_file_write(handle, offset,
    segment, nbytes, result);
```

REFERENCE

MINDSET MS-DOS Programmers Reference Manual, Section 1, Read from File.

DESCRIPTION

Dos_file_write issues a request to MS-DOS to write the specified number of bytes to the given file handle. The bytes to be written are stored in the buffer pointed to by segment and offset.

Issuing a dos_file_write with the number of bytes set to 0 will set the file size to the current position (update the file size).

CAUTIONS

Registers AX,BX,CX,DX,DS in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 21
Value of (AH)= 41

dos_file_delete - Delete the specified file from the specified path.

SYNOPSIS

--- C ---

```
int offset;   Pointer to source buffer
int segment;  Segment of source buffer
int result;   Result of MS-DOS operations
                0 - failure
               -2 - file not found
               -5 - access denied

result = dos_file_delete(offset,
           segment;
```

--- PASCAL ---

```
offset: word;
segment: word;
var result: integer;

dos_file_delete(offset,
  segment, result);
```

REFERENCE

MINDSET MS-DOS Programmers Reference Manual, Section 1, Delete a Directory Entry.

DESCRIPTION

Dos_file_delete issues a request to MS-DOS to delete the specified file from the specified directory and path.

CAUTIONS

Registers AX,BX,CX,DX,DS in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = 21
Value of (AH) = 42

dos_file_lseek - Position the read/write pointer in the file with the specified handle.

SYNOPSIS

--- C ---

```
int handle;  File handle for file
int method;  Method used for positioning
                   0 - Absolute move to offset
                   1 - Relative move from current
                   2 - Move to end of file plus offset

int lo_offset; Low 16 bits of offset
int hi_offset; High 16 bits of offset
int result;  Result of MS-DOS operations
                  0 - failure
                 -1 - invalid function
                 -6 - invalid handle

result = dos_file_lseek(handle, method,
          lo_offset, hi_offset);
```

--- PASCAL ---

```
handle: integer;
method: integer;

offset: word;
segment: word;
var result: integer;

dos_file_delete(offset, method,
 lo_offset, hi_offset, result);
```

REFERENCE

MINDSET MS-DOS Programmers Reference Manual, Section 1, Delete a Directory Entry.

DESCRIPTION

Dos_file_lseek issues a request to MS-DOS to position the read/write pointer in the file from the specified directory and path.

Lo_offset and hi_offset constitute a 32-bit integer for specifying the total number of bytes offset the pointer is to be moved.

CAUTIONS

Registers AX,BX,CX,DX,DS in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

Interrupt # = EE
Value of (AH) = 02

NAME

set_display_device - Set display device to television (composite) or monitor (RGB) and reloads the color palette.

SYNOPSIS

--- C ---

char device_code; value for device code

--- PASCAL ---

device_code: byte;

device_ code	device	color/B&W
0	television	color
1	monitor	N/A
2	television	B&W
3	monitor	N/A

set_display_device(device_code);

REFERENCE

MINDSET SDG Section 4 (Descriptions of Display Processor BIOS commands).

DESCRIPTION

This routine changes the current display setup for either a color monitor/television or a black and white television (B&W). The color palette is reloaded (according to current display mode - see Reference) according to the device selected.

CAUTIONS

Register AL in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

AME

Interrupt # = EE
Value of (AH) = 06

set_sync_mode - Enables or disables the use of genlock for transparent colors; enables or disables interlaced sync display, and enables or disables fixed-phase display.

SYNOPSIS

--- C ---

```
char sync_mode;
```

bits set for desired sync

--- PASCAL ---

```
sync_mode: byte;
```

bit	value	features
0	0	Disable genlock
	1	Enable genlock
1		Unused
2	0	Disable interlaced sync
	1	Enable interlaced sync
3	0	Disable fixed phase
	1	Enable fixed phase

```
set_sync_mode(sync_mode);
```

REFERENCE

MINDSET SDG Section 4 (Display Interrupt Control and VBLANK Operations).

DESCRIPTION

Set_sync_mode enables the user to enable and disable certain video signal inputs.

Genlock is normally disabled. When enabled, those palette entries with the key bit set (Refer to set_palette, Interrupt # EF, Value of (AH) = 0A) are transparent, allowing a second video signal to be displayed.

Interlaced sync display is normally disabled. When enabled, this mode displays 200 scan lines of display data, but displays using an even/odd scan line pair to display 400 lines of data for a more filled-in look on the display.

Fixed-phase synchronization is normally disabled. When enabled, it gives more flexibility in pixel-by-pixel color mixing and prevents flashing on the display.

See Mindset SDG Section 4, Descriptions of Display Processor BIOS Command.

CAUTIONS

Enabling fixed-phase synchronization may cause the television display to be distorted.

Registers AL and BL in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EE
Value of (AH) = 07

set_display_int - Specifies the scan line at which the read processor performs an interrupt. This command also enables and disables a diagnostic marker which appears on the specified scan line.

SYNOPSIS

--- C ---

```
char linenum;   decimal value of scan line
                where interrupt occurs (0 - 399)

                set_display_int(linenum);
```

--- PASCAL ---

```
linenum: byte;
```

REFERENCE

MINDSET SDG Section 4 (Display Interrupt Control and VBLANK Operations).

DESCRIPTION

Set_display_int enables the user to synchronize animation with the drawing of the display screen by the display processor.

To enable the diagnostic marker, insert the following line just prior to the set_display_int call;

```
    .
    .
  myregs.bx = 1;
  set_display_int(linenum);
    .
    .
```

See Mindset SDG Section 4, Descriptions of Display Processor BIOS Commands, Set Display Interrupt Address (Interrupt # = EF, Value of (AH) = 0F).

CAUTIONS

The display interrupt operates only in the graphics modes of the MINDSET computer.

If no value for linenum is given, then a default value of 199 (decimal) is used.

Registers AL and BL in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EE
Value of (AH) = 1F

joystick - Returns the position and buttons status of two joystick/mouse devices.

SYNOPSIS

--- C ---

```
char *a_switch, *b_switch;
```

--- PASCAL ---

```
var a_switch: byte;
var b_switch: byte;
```

joystick switch variables contain button sense (pushed/not-pushed), where pushed buttons have a bit value of 0, not-pushed buttons are 1.

Joystick switches bitmap

bit #	meaning
0	up direction switch
1	down direction switch
2	left direction switch
3	right direction switch
4	button #1 (left side)
5	button #2 (right side)

```
int *a_x, *a_y, *b_x, *b_y;
```

```
var a_x,a_y: integer;
var b_X,b_y: integer;
```

(X,Y) position change for device a,b

```
joystick(a_switch, b_switch, a_x, a_y, b_x, b_y);
```

REFERENCE

MINDSET SDG Section 10 (Miscellaneous BIOS Commands).

DESCRIPTION

Joystick enables the use of 1 or 2 joystick/mouse devices as input.

Variables a_switch and b_switch contain bits corresponding to switches indicating the direction of movement of the joystick.

Variables a_x, a_y, b_x, b_y contain the X-axis and Y-axis values for joysticks with decoder inputs. These X and Y values reflect the change in each axis since the last time this routine was called (relative positioning).

CAUTIONS

Registers AX,BX,CX,DX,SI in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EE
Value of (AH) = 22

set_led - Turns the MINDSET computer front panel LED's on/off.

SYNOPSIS

--- C ---

```
char on_off; bit pattern for setting LED's
```

--- PASCAL ---

```
on_off: byte;
```

bit# 1 0	value	yellow -- LED -- green
0 0	0	off off
0 1	1	on off
1 0	2	off on
1 1	3	on on

```
set_led(on_off);
```

REFERENCE

MINDSET SDG Section 10 (Miscellaneous BIOS Commands).

DESCRIPTION

The front betzel of the MINDSET computer base unit (with power cord) contains two user-programmable Light Emitting Diodes (LED). These LED's are both located just right of the center of the betzel.

The left-hand LED is yellow and is designated LED 0. LED 0 is turned on/off by setting the value of bit 0 in the on_off variable to 1 or 0 respectively.

The right-hand LED is green and is designated LED 1. LED 1 is turned on/off by setting the value of bit 1 in the on_off variable to 1 or 0 respectively.

LED 0 and 1 remain in the state set by the last call to set_led.

CAUTIONS

Register AL in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EE
Value of (AH) = 24

sound_mode - Sets operation mode of 1 or 2 Custom Sound Processors(CSP)

SYNOPSIS

--- C ---

char mode; CSP mode from table (below)

CSP mode	type of sound
1	4 musical voices, some special effects
2	3 voices, special music/noise effects
4	6 voices with limited controls
8	Direct access to digital-analog (D/A) converter

char module_bits; Select on-board or stereo (module) mode

CSP module	CSP enabled
1	On-board CSP
2	Module CSP (in optional stereo module)
3	Both CSP's

sound_mode(mode, module_bits);

--- PASCAL --

mode: byte;

module_bits: byte;

REFERENCE

MINDSET SDG Section 5 (Custom Sound Processor - CSP Operation).

DESCRIPTION

Sound_mode selects the mode of sound output from 1 or 2 CSP's.

The MINDSET computer base unit has one on-board CSP. The second CSP, required for stereo modes, is a module which inserts into the MINDSET computer.

CAUTIONS

Register AL in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

Interrupt # = EE
Value of (AH)= 25

NAME

sound_regs - Sets the registers that control CSP operation.

SYNOPSIS

--- C ---

char module_bits; Select on-board or stereo (module) mode

CSP module	CSP enabled
1	On-board CSP
2	Module CSP (in optional stereo module)
3	Both CSP's

--- PASCAL ---

module_bits: byte;

int mask1, mask2;
Bit patterns select which sound registers may be written into (mask1 for on-board CSP, mask2 for stereo CSP)

mask1, mask2 : integer;

int soundtable;
Table of which CSP registers are affected by bits in mask1 and mask2. (See SDG Section 5, for table in description of SET SOUND REGISTERS).

soundtable: integer;

sound_regs(module_bits, mask1, mask2, soundtable);

REFERENCE

MINDSET SDG Section 5 (Custom Sound Processor - CSP Operation).

DESCRIPTION

Sound_regs selects which CSP registers are to be altered, based on user supplied mask1 (and mask2 for stereo CSP).

The MINDSET computer base unit has one on-board CSP. The second CSP, required for stereo modes, is a module which inserts into the MINDSET computer.

CAUTIONS

Registers AL,BX,CX,ES,SI in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EE
Value of (AH) = 26

sound_data - Transfers sound data directly from the 80186 CPU to the digital-analog (D/A) converter.

SYNOPSIS

--- C ---

char module_bits; Select on-board or stereo (module) mode

CSP module	CSP enabled
1	On-board CSP
2	Module CSP (in optional stereo module)
3	Both CSP's

--- PASCAL ---

module_bits: byte;

int data1, data2; Sound data for on-board CSP (data1) and stereo CSP (data2).

--- PASCAL ---

data1, data2: integer;

```
sound_data(module_bits, data1, data2);
```

REFERENCE

MINDSET SDG Section 5 (Custom Sound Processor - CSP Operation).

DESCRIPTION

Sound_data writes data directly to the D/A converter, from the 80186 CPU in the MINDSET computer. Two data bytes (8 bits/byte) may be written to the CSP(s).

The MINDSET computer base unit has one on-board CSP. The second CSP, required for stereo modes, is a module which inserts into the MINDSET computer.

CAUTIONS

This routine only works in see-through mode (mode 4). See sound_mode routine description (Interrupt # = EE, Value of (AH) = 24).

Registers AL and BX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

Interrupt # = EE
Value of (AH) = 27

NAME

stereo_check - Checks for the presence of a stereo module.

SYNOPSIS

--- C ---

```
char stereo;  Flag set to 2 if stereo
              module is present

stereo  = stereo_check();
```

--- PASCAL ---

```
stereo: boolean;

stereo_check(stereo);
```

REFERENCE

MINDSET SDG Section 5 (Custom Sound Processor - CSP Operation).

DESCRIPTION

Stereo_check enables the calling program to determine if a second (optional) CSP is present in the MINDSET computer.

The MINDSET computer base unit has one on-board CSP. The second CSP, required for stereo modes, is a module which inserts into the MINDSET computer.

CAUTIONS

Register AL in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EE
Value of (AH)= 36

enable_beeper - Enables or disables the system beeper.

SYNOPSIS

--- C ---

```
char on_off; If onoff = 0 then disable beeper
               onoff = 1 then enable beeper

             enable_beeper(on_off);
```

--- PASCAL ---

```
on_off: byte;
```

REFERENCE

MINDSET SDG Section 10 (Miscellaneous BIOS Commands).

DESCRIPTION

Enable_beeper routine controls beeper operation.

If enabled, the system beeper will produce an audible sound ('beep') in response to any of the following conditions:

1) A set_beeper routine call is made,
2) A bell character (usually CTRL-G) is generated by write_teletype routine (Interrupt # = 10, Value of (AH) = 0E),
3) A CTRL-G is pressed on the keyboard,
4) The keyboard buffer becomes full (approx. 15 keystrokes).

If a television is used for display device, or the system audio output is connected, then the system beeper need not be used. Each of the above 4 conditions will cause the television or device connected to the audio output, to produce a beep. The system audio output is the output of the on-board Custom Sound Processor (CSP) at the jack on the rear panel of the base unit labelled AUDIO LEFT. See SDG Section 1 (An Architectural Overview).

CAUTIONS

Register AL in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EE
Value of (AH)= 37

test_beeper - Returns beeper enabled/disabled status.

SYNOPSIS

--- C ---	--- PASCAL ---
char enabled; Beeper disabled if = 0, enablerd if = 1	**enabled: boolean;**
enabled = test_beeper();	**test_beeper(enabled);**

REFERENCE

MINDSET SDG Section 10 (Miscellaneous BIOS Commands).

DESCRIPTION

Test_beeper returns the current enabled/disabled status of the system beeper. See enablebeeper (Interrupt # = EE, Value of (AH) = 36), or the MINDSET SDG Section 10 (Miscellaneous BIOS Commands).

If enabled, the system beeper will produce an audible sound ('beep') in response to any of the following conditions:

1) A set_beeper routine call is made,
2) A bell character (usually CTRL-G) is generated by write_teletype routine (Interrupt # = 10, Value of (AH) = 0E),
3) A CTRL-G is pressed on the keyboard,
4) The keyboard buffer becomes full (approx. 15 keystrokes).

If a television is used for display device, or the system audio output is connected, then the system beeper need not be used. Each of the above 4 conditions will cause the television or device connected to the audio output, to produce a beep. The system audio output is the output of the on-board Custom Sound Processor (CSP) at the jack on the rear panel of the base unit labelled AUDIO LEFT. See SDG Section 1 (An Architectural Overview).

CAUTIONS

Register AL in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EE
Value of (AH)= 38

set_beeper - Turns the beeper sound on/off, if the beeper is enabled.

SYNOPSIS

--- C ---

```
char on_off;  If on_off = 0 then turn beeper off
                 on_off = 1 then turn beeper on
                          (if beeper enabled)

              set_beeper(on_off);
```

--- PASCAL ---

```
on_off: byte;
```

REFERENCE

MINDSET SDG Section 10 (Miscellaneous BIOS Commands) and enablebeeper (Interrupt # = EE, Value of (AH) = 36).

DESCRIPTION

Set_beeper turns the system beeper on or off, providing that the beeper is enabled (by enable_beeper - Interrupt # EE, Value of (AH) = 36H).

The system beeper sounds from the time set_beeper turns it on, until set_beeper turns it off.

The set_beeper routine has no effect if the beeper is disabled. See references for additional details.

CAUTIONS

Register AL in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 00

set_screen_mode -- Set mode of video display

SYNOPSIS

--- C ---

char mode; value for video mode

--- PASCAL ---

mode: byte;

Mode table:

mode	resolution	colors	buffer(s)
0	320 x 200	2	1 or 2
1	320 x 200	4	1 or 2
2	320 x 200	16	1 only
3	640 x 200	2	1 or 2
4	640 x 200	4	1 only
5	320 x 400	4	1 only
6	640 x 400	2	1 only

setmode(mode);

REFERENCE

Reference MINDSET SDG Section 4 (under Display Processor BIOS Commands).

DESCRIPTION

This routine controls only the MINDSET graphics display modes. INT 10 sets IBM-compatible screen modes for the MINDSET. Modes 0,1,3 may be used with single or double buffering.

Invoking set_screen_mode causes certain functions to be performed. These are listed in the SDG in section 6 under Screen Mode Commands.

The current screen mode may be obtained by calling get_screen_mode (Interrupt # EF, Value of (AH) = 01).

CAUTIONS

The mode for parameter blocks is set to contiguous (see Reference).

No checking of the value of mode is performed. Unpredictable results may occur if mode is not within the specified range.

The user must set collision/clip/transparency/transfer mode parameters as desired (see Reference).

Register AL in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
1.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value in (AH)= 01

get_screen_mode - Returns current screen mode parameters

SYNOPSIS

--- C ---		--- PASCAL ---
int *flags;	System video function flags	**var flags: integer;**
char mode;	Display mode (see set_screen_mode)	**var mode: byte;**
char *bitspix;	Number of bits per pixel (# colors = 2** bits/pixel)	**var bits_per_pixel:byte;**
mode = get_screen_mode(flags,bitspix);		**get_screen_mode(flags, mode, bits_per_pixel);**

REFERENCE

Reference MINDSET SDG chapters 3 and 6.

DESCRIPTION

Three parameters are returned; a flag variable containing 16 flag bits to reflect the current status of the display mode, the value of mode from the most recent set_screen_mode, and the current number of bits per pixel.

Detailed flag descriptions may be found in the SDG in chapter 6, under Screen Mode commands.

A table of mode values may be found in the SDG in chapter 6, under Screen Mode commands, or in this document under set_screen_mode (Interrupt # EF, Value of (AH) = 00).

The third parameter returned is the current number of bits/pixel used for the screen display.

CAUTIONS

The values of 80186 registers AX, BX and CX are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value in (AH)= 02

set_transfer_mode - Specifies how the GCP modifies the destination data with the source data

SYNOPSIS

--- C ---

```
int mode;          Entry in xfermode table
```

Xfermode table:

------ xfermode ------		
opaque	transparent	logical combination mode
0	255	Move source into destination
1	256	AND source into destination
2	257	OR " " "
3	258	XOR " " "
4	259	NOT source and replace dest
5	260	AND (NOT source) into dest
6	261	OR (NOT source) into dest
7	262	XOR (NOT source) into dest

--- PASCAL ---

```
mode: integer;
```

```
set_transfer_mode(mode);
```

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under GCP commands).

DESCRIPTION

This routine performs two functions;
1) selects visibility mode (opaque or transparent) for pixel display
2) determines how source pixels are combined with destination pixels

Opaque visibility transfers all source pixels for combination with the destination pixels. Transparent visibility transfers only those source pixels which are non-zero for combination with destination pixels. For non-zero pixels, transparent and opaque visibility are identical.

The logical transfer modes determine how the data from each source pixel is combined with data from each destination pixel. The table (above) lists the various logical combinations of source and destination pixels. Transparency applies only to pixels which are not zero BEFORE the NOT operation.

CAUTIONS

Visibility mode is set to opaque, and the logical transfer mode set to replace all destination pixels with source pixels when set_transfer_mode is invoked with no argument.

Registers AX and BX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value in (AH)= 03

get_transfer_mode - Get current visibility and logical transfer modes.

SYNOPSIS

--- C ---

```
int mode;    Value of current mode
             (see set_transfer_mode table)

mode = get_transfer_mode();
```

--- PASCAL ---

```
var mode: integer;

get_transfer_mode(mode);
```

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under GCP commands).

DESCRIPTION

This routine returns the current selection of the following functions;
1) visibility mode (opaque or transparent) for pixel display
2) logical combination mode of source and destination pixels

The value returned by the routine corresponds to the transfer mode table (see set_transfer_mode - Interrupt # EF, Value of (AH) = 02 in this document, or chapter 6 in the SDG).

CAUTIONS

Registers AX and BX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value in (AH) = 04

set_dest_buffer - Specifies address and size of destination buffer for all subsequent GCP operations

SYNOPSIS

--- C ---		--- PASCAL ---
int offset;	Offset of destination buffer	**offset: word;**
int segment;	Segment of destinatin buffer	**segment: word;**
int width;	Number of bytes per scan line (must be even number)	**width: integer;**
int lines;	Number of scan lines in destination buffer, a scan line is one pixel high	**lines: integer;**

setdest(offset, segment, width, lines);

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under GCP commands).

DESCRIPTION

This routine prepares the system for GCP operations by defining the destination buffer for all subsequent data transfers originating from any source buffer.

The destination buffer may reside in any segment of the 1 megabyte address space of the 80186, provided the buffer begins on an even (word, not byte) boundary.

CAUTIONS

This routine sets the clip rectangle to match the bounds of the new destination buffer.

Registers AX,BX,CX,DX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value in (AH)= 05

get_dest_buffer - Get current address and size of destination buffer

SYNOPSIS

--- C ---		--- PASCAL ---
int *offset;	Offset of destination buffer	**var offset: word;**
int *segment;	Segment of destination buffer	**var segment: word;**
int *width;	Number of bytes per scan line (must be even number)	**var width: integer;**
int *lines;	Number of scan lines in destination buffer, a scan line is one pixel high	**var lines: integer;**

get_dest_buffer(offset, segment, width, lines);

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under GCP commands).

DESCRIPTION

This routine returns the current address and size of the destination buffer.

The destination buffer is set by set_dest_buffer (Interrupt # EF, Value of (AH) = 04).

CAUTIONS

Registers AX,BX,CX,DX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value in (AH)= 06

set_write_mask - Sets the write mask for all subsequent GCP operations

SYNOPSIS

--- C ---

int mask; 16 bit (1 word) defining write mask;
if bit = 0 then not modified by GCP.
if bit = 1 then allow modify by GCP

set_write_mask(mask);

--- PASCAL ---

mask: word;

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under GCP commands).

DESCRIPTION

This routine defines a 16 bit mask for use in subsequent GCP operations. The GCP examines this mask before transferring data to the destination buffer. The bit positions of the write mask correspond to the bit positions of the words from the source buffer.

A mask of FFFFH (hexadecimal) allows all source word bits to be transferr to the destination buffer (with modification by the GCP - see routine set_transfer_mode (Interrupt # EF, Value of (AH) = 02).

A mask of 0000H prevents the GCP from modifying any source word bits during the transfer to the destination buffer.

CAUTIONS

The routine set_transfer_mode (Interrupt # EF, Value of (AH) = 02) sets the write mask to FFFFH.

Registers AX and BX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH) = 07

get_write_mask - Returns the current write mask for GCP operations

SYNOPSIS

--- C ---		--- PASCAL ---
`int mask;`	Mask containing bits of write mask (see set_write_mask)	`var mask: word;`
`mask = get_write_mask();`		`get_write_mask(mask);`

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under GCP commands).

DESCRIPTION

Value of routine is 16 bit word containing the current write mask to be used for GCP operations.

The write mask is selected with set_write_mask (Interrupt # EF, Value of (AH) = 06).

CAUTIONS

Registers AX and BX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value in (AH)= 08

blt_copy - GCP copies data block from source to destination buffer

SYNOPSIS

--- C ---

```
char id;          User assigned reference number      id: byte;
                  GCP identification
int count;        Number of blts GCP is to perform   count: integer;
int mode;         S - source, D- destination,        mode: integer;
                  X - both buffers
```

--- PASCAL ---

BIT#	0/1	READ	WRITE	TOP	BOTTOM	LEFT	RIGHT	FAST BLT
0	0	S		S	-> S			
	1	S		S	<- S			
1	0		D	D	-> D			
	1		D	D	<- D			
2	0	X	X			X	-> X	
	1	X	X			X	<- X	
3	0	- same as bit 2 -						
	1	- same as bit 2 -						
4	0							normal
	1							fast

```
int xorg,yorg;  (X,Y) coordinate in destination   xorg, yorg: integer;
                buffer to begin copying source data
int offset;     Offset of (0,0) pixel in source   offset: word;
                buffer
int segment;    Segment of (0,0) pixel in source  segment: word;
                buffer
```

```
blt_copy(id, count, direction, xorigin, yorigin, offset);
```

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under GCP commands).

DESCRIPTION

The blt_copy routine specifies a series of block transfer operations frc one or more source buffers to a common destination buffer. The object definitions are stored as an array of parameter groups in memory. The user must supply a **blt_copy parameter group** for each separate object to be transferred.

The id argument is used by the GCP in reporting collision/clip detection. Count determines the number of parameter groups to be transferred by this blt_copy call. Mode indicates the orientation of the transfer (see table). Xorg and yorg provide a common point of reference within the destination buffer for block transfers specified by a single blt_copy. Pixel location (0,0) is the upper left corner of the display, with positive x-coordinates to the left, and positive y-coordinates down towards the bottom of the display.

The **blt_copy parameter group** is defined as follows:

Byte offset		Parameter
+ 0		Source address offset
+ 2		Source address paragraph (segment)
+ 4		Source width in bytes (must be an even number)
+ 6		X-source offset from source address in pixels
+ 8		Y-source offset from source address in pixels
+ 10		X-destination offset from x-origin in pixels
+ 12		Y-destination offset from y-origin in pixels
+ 14		X-size width of source buffer region in pixels
+ 16		Y-size height of source buffer region in pixels
+ 18		Source mask to AND with source data during bltcopy
+ 20	(optional)	Pointer to next parameter group (linked list) (Refer to setlink for definition of this pointer)

Additional detail on the blt_copy may be found in the MINDSET Software Developer Guide (SDG) in chapter 6 (under GCP commands).

CAUTIONS

Note the correct set-up sequence for blt_copy in the SDG (example below).

Refer to the SDG (bltcopy) for restrictions on the use of fast blt.

Registers AX,BX,CX,DX,SI,DI,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 09

blt_copy_word - GCP fills rectangular regions of destination buffer with a 16-bit pixel pattern

SYNOPSIS

--- C ---		--- PASCAL ---
char id;	User assigned reference number for GCP identification	**id: byte;**
int count;	Number of blts GCP is to perform	**count: integer;**
int xorg,yorg;	(X,Y) coordinate in destination buffer to begin copying source data	**xorg,yorg: integer;**
int offset;	Address of (0,0) pixel in source buffer	**offset: word;**
int segment;	Segment of (0,0) pixel in source buffer	**segment: word;**

blt_copy_word(id, count, xorigin, yorigin, offset);

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under GCP commands).

DESCRIPTION

Each blt_copy_word specifies a series of block fill operations within the destination buffer. The definitions of the regions to be filled and the fill patterns are stored as an array of parameter groups in memory. The user must supply a **blt_copy_word parameter group** for each filled block in the destination buffer.

The **blt_copy_word parameter group** is defined as follows:

Byte offset		Parameter
+ 0		Fill pattern word
+ 2		X-destination offset from x-origin in pixels
+ 4		Y-destination offset from y-origin in pixels
+ 6		X-size width of filled rectangle in pixels
+ 8		Y-size height of filled rectangle in pixels
+ 10	(optional)	Pointer to next parameter group (linked list) (Refer to setlink for definition of this pointer)

Additional detail on blt_copy_word may be found in the MINDSET Software Developer Guide (SDG) in chapter 6 (under GCP commands).

CAUTIONS

Note the correct set-up sequence for blt_copy_word in the Mindset SDG (under blt_copy).

Registers AX,BX,CX,SI,DI,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH) = 0A

set_palette - Specifies the contents of the color palette and selects the color of the screen border

SYNOPSIS

--- C ---		--- PASCAL ---
`char border;`	Color palette index to be used for display border	`border: byte;`
`int index;`	Index in color palette to begin setting new colors	`index: integer;`
`int count;`	Number of palette color entries to be set	`count: integer;`
`int offset;`	Offset of new color palette	`offset: word;`
`int segment;`	Segment of new color palette	`segment: word;`

```
set_palette(border, index, count, offset, segment);
```

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under color palette commands).

DESCRIPTION

The set_palette routine specifies all or part of the contents of the color palette and selects the display screen border color. The user must supply a color data array in memory which set_palette copies into the color palette.

Definition of color palette entry:

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Device:	M	M	M	M	0	0	T	T	T	T	T	T	T	T	T	T
Color(key):	I	R	G	B	0	0	K	B	B	B	G	G	G	R	R	R

M - monitor color bit T - television color bit 0 - unused
R - red G - green B - blue
I - intensity K - key onto external video signal

Additional detail on set_palette may be found in the MINDSET Software Developer Guide (SDG) in chapter 6 (under color palette commands).

Refer to the get_palette (Interrupt # EF, Value of (AH) = 0BH) routine for reading the currently set palette.

CAUTIONS

Note that correct palette color indices depend on the current screen mode (refer to SDG Chapter 6 under color palette commands).

Registers AX,BX,CX,DX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH) = 0B

get_palette - Returns the contents of the color palette and current color of the screen border

SYNOPSIS

--- C ---		--- PASCAL ---
char *border;	Color palette index used for display border	var border: byte;
int index;	Index in color palette to begin getting entries	index: integer;
int count;	Number of palette color entries to get	count: integer;
int offset;	Offset to data array to store color palette	offset: word;
int segment;	Segment of data array to store color palette	segment: word;

```
get_palette(border, index, count, offset, segment);
```

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under color palette commands).

DESCRIPTION

Get_palette returns all or part of the contents of the color palette and the index of into the color palette which selects the current screen border color. The user must provide an area in memory to receive the color data from the color palette.

Additional detail on get_palette may be found in the MINDSET Software Developer Guide (SDG) in chapter 6 (under color palette commands).

Refer to set_palette (Interrupt # EF, Value of (AH) = 0AH) for writing a new palette selection, and definition of each palette entry.

CAUTIONS

Note that correct palette color indexes depend on screen mode (refer to SDG chapter 6 under color palette commands).

Registers AX,BX,CX,DX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 0C

blt_polypoint - Draws a collection of points of the same color into the buffer

SYNOPSIS

--- C ---		--- PASCAL ---
`char id;`	User assigned reference number for this blt	`id: byte;`
`int count;`	Number of points to be drawn	`count: integer;`
`char color;`	Color palette index for this collection of points	`color: byte;`
`int xorg,yorg;`	Coordinates in the destination buffer at which to start drawing points	`xorg, yorg: integer;`
`int offset;`	Offset of coordinate list	`offset: word;`
`int segment;`	Segment of coordinate list	`segment: word;`

```
points(id, count, color, xorigin, yorigin, offset, segment);
```

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under draw commands).

DESCRIPTION

Blt_polypoint draws a collection of points at locations which the user specifies as a list of coordinates. All points specified in the coordinate set are drawn using the same color palette index.

Each entry in the coordinate set is defined as follows:

Byte offset	Parameter
+ 0	X-coordinate (signed offset) in pixels
+ 2	Y-coordinate (signed offset) in pixels

Additional detail on blt_polypoint may be found in the MINDSET Software Developer Guide (SDG) in chapter 6 (under draw commands).

CAUTIONS

Note that the number of points (count) to be drawn should not exceed 16 if the user intends to use the GCP blt_wait routine to poll for collision and clipping information.

Note that correct palette color indices depend on screen mode (refer to SDG chapter 6 under color palette commands).

Registers AX,BX,CX,DX,ES,DI,SI in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH) = 0D

blt_polyline - Draws a collection of straight lines of the same color into the destination buffer

SYNOPSIS

--- C ---		--- PASCAL ---
`char id;`	User assigned reference number for this blt	`id: byte;`
`char mode;`	Draw mode: 0 - chained, 1 - linked lines	`mode: polyline_mode;`
`int count;`	Number of lines to be drawn	`count: integer;`
`char color;`	Color palette index for this collection of lines	`color: byte;`
`int xorg,yorg;`	Coordinates in the destination at which to start drawing lines	`xorg, yorg: integer;`
`int offset;`	Offset to first entry of coordinate set	`offset: word;`
`int segment;`	Segment of first entry of coordinate set	`segment: word;`

```
blt_polyline(id, mode, count, color, xorg, yorg, offset, segment);
```

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under draw commands).

DESCRIPTION

Blt_polyline draws a collection of lines at locations which the user specifies as a list of coordinates. All lines specified in the coordinate set are drawn using the same color palette index.

The value of the mode flag determines whether the **coordinate set** is interpreted as pairs of points (for line segments), or continuous points (for linked lines).

Each entry in the **coordinate set** is defined as follows:

Byte offset	Parameter
+ 0	X-coordinate (signed offset) in pixels
+ 2	Y-coordinate (signed offset) in pixels

Additional detail on lines may be found in the MINDSET Software Developer Guide (SDG) in chapter 6 (under draw commands).

CAUTIONS

Note that correct palette color indices depend on screen mode (refer to SDG chapter 6 under color palette commands).

Registers AX,BX,CX,DX,ES,DI,SI in the 80186 are altered by this routine

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 0E

get_buffer_info - Returns the addresses and size of the system frame buffer

SYNOPSIS

--- C ---		--- PASCAL ---
int *fb1_segment;	Frame buffer 1 segment	**var fb1_segment: word;**
int *fb2_segment;	Frame buffer 2 segment	**var fb2_Segment: word;**
int *size;	Size of each buffer in bytes	**var size: integer;**

get_buffer_info(fb1_segment, fb2_segment, size);

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under frame buffer commands).

DESCRIPTION

The system provides 2 frame buffers for most display modes, to allow enhanced display switching via double buffering. In display modes where double buffering is not possible (320x200x4 bits/pixel), fb1_segment is the address of the single system frame buffer with an actual number of bytes = 2 * size.

Frame buffers are always paragraph aligned.

Additional detail on get_buffer_info may be found in the MINDSET Software Developer Guide (SDG) in chapter 6 (under frame buffer commands).

CAUTIONS

In some cases,the two frame buffers are not contiguous. Therefore, do not use the difference between the two buffer addresses to calculate the buffer size.

Registers AX,BX,CX,DX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	06/01/84	Original version.

Interrupt # = EF
Value of (AH) = 0F

NAME

set_disp_int_addr - Specifies the address of the user-defined display interrupt routine

SYNOPSIS

--- C ---		--- PASCAL ---
int offset;	Offset of user interrupt service routine	**offset: integer;**
int segment;	Segment of user interrupt service routine	**segment: integer;**

set_disp_int_addr(offset);

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under frame buffer commands).

DESCRIPTION

Set_disp_int_addr specifies the segment and offset address of the user-defined interrupt service routine. The system calls this routine each time it finishes writing the frame buffer image to the screen, or when the GCP reaches the scan line selected by set_display_int (Interrupt # EE, Value of (AH) = 07).

Specifying an address of 0 informs the system that there is no user-defined interrupt routine service.

The last scan line on the screen is the default scan line for the display interrupt. This scan line conincides with the vertical blanking (VBLANK) signal. In this case, the user's interrupt routine has approximately 1 millisecond before the flyback signal occurs. The system interrupts the user's routine at this time to perform its' late VBLANK procedures, and then returns to the user's routine.

Additional detail on set_disp_int_addr may be found in the MINDSET Software Developer Guide (SDG) in chapter 6 (under frame buffer commands).

Refer to get_disp_int_intaddr (Interrupt # EF, Value of (AH) = 10H) for obtaining the current address of the user interrupt service routine.

CAUTIONS

The user program must terminate the display interrupt vector (with offset of 0) before terminating operation. Otherwise, the next display interrupt will cause the system to transfer to a non-existent service routine.

Registers AX,BX,CX,DX in the 80186 are altered by this routine.

REVISION

Version	Date	Comment
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH) = 10

get_disp_int_addr - Returns the address of the user-defined display interrupt service routine

SYNOPSIS

--- C ---		--- PASCAL ---
int *offset;	Offset of user interrupt routine	**var offset: word;**
int *segment;	Segment of user interrupt rountine	**var segment: word;**

get_disp_int_addr(offset, segment);

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under frame buffer commands).

DESCRIPTION

Get_disp_int_addr returns the paragraph and offset addresses of a user-defined interrupt service routine set by set_disp_int_addr (Interrupt # EF, Value of (AH) = 0FH).

If the segment and offset returned are 0, then there is no current user-defined interrupt service routine.

Additional detail on get_disp_int_addr may be found in the MINDSET SDG, Chapter 6 (under frame buffer commands).

Refer to set_disp_int_addr for setting the address of a user-defined interrupt service routine.

CAUTIONS

The user program must terminate the display interrupt vector (with offset of 0) before terminating operation. Otherwise, the next display interrupt will cause the system to transfer to a non-existent service routine.

Registers AX,BX,CX,DX in the 80186 are altered by this routine.

REVISION

Version	Date	Comment
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 11

switch_active_buffer - Causes the system to switch active frame buffers

SYNOPSIS

--- C ---

```
switch_active_buffer();
```

--- PASCAL ---

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under frame buffer commands).

DESCRIPTION

Switch_active_buffer operates in display modes with double buffering to switch the active frame buffer with the hidden (not currently displayed) frame buffer. See set_screen_mode (Interrupt # EF, Value of (AH) = 00) for those display modes which have double buffering.

This routine has no effect in modes restricted to single buffering, such as 320x200x4 bits/pixel. Refer to set_screen_mode (Interrupt # EF, Value of (AH) = 00H) for a description of display modes.

Additional detail on switch_active_buffer may be found in the MINDSET SDG Chapter 6 (under frame buffer commands).

CAUTIONS

Register AX in the 80186 is altered by this routine.

REVISION

Version	Date	Comment
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 12

set_collision_pattern - Defines the criteria for collision detection by the GCP

SYNOPSIS

--- C ---		--- PASCAL ---
int polarity;	0 - disable, 1 - enable all-except mode	**polarity: integer;**
char bit_pattern;	Bit pattern for collision detection	**bit_pattern: byte;**
int mask;	Defines the bitmask for don't care bits	**mask: byte;**

set_collision_pattern(polarity, bit_pattern, mask);

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under collision and clip commands).

DESCRIPTION

Set_collision_pattern specifies a bit pattern which, when matched, shows that the preceeding GCP BLT operation caused a collision. The bit pattern is compared with each pixel as it is copied into the destination buffer. The number of bits in the pattern is equal to the number of bits in a single pixel in the current display mode.

The all_except mode specification reverses the meaning of the pattern/mask, indicating a collision on anything except pixels matching the pattern/mask.

The mask parameter includes a 0 bit for each bit position in the pattern which the system should ignore when checking for collisions. A value of 0 for all bits in the mask causes the system to detect a collision after every BLT operation.

Additional detail on set_collision_pattern may be found in the MINDSET SDG, Chapter 6 (under collision and clip commands).

Refer to get_collision_pattern (Interrupt # EF, Value of (AH) = 13H) for determining the current collision detect criteria.

CAUTIONS

Registers AX,BX in the 80186 are altered by this routine.

REVISION

Version	Date	Comment
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 13

get_collision_pattern - Returns the criteria for collision detection by the GCP

SYNOPSIS

--- C ---		--- PASCAL ---
int *polarity;	0 - disable 1 - enable all-except mode	**var polarity: integer;**
char *pattern;	Bit pattern for collision detection	**var pattern: byte;**
int *mask;	Defines the bitmask for don't care bits	**var mask: integer;**

get_collision_pattern(polarity, pattern, mask);

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under collision and clip commands)

DESCRIPTION

Get_collision_pattern returns the bit pattern which, when matched, shows that the preceeding GCP BLT operation caused a collision. The bit patter is compared with each pixel as it is copied into the destination buffer. The number of bits in the pattern is equal to the number of bits in a single pixel in the current display mode.

Additional detail on get_collision_pattern may be found in the MINDSET SD chapter 6 (under collision and clip commands).

Refer to set_collision_pattern (Interrupt # EF, Value of (AH) = 12H) for defining the collision detect criteria.

CAUTIONS

Registers AX,BX in the 80186 are altered by this routine.

REVISION

Version	Date	Comment
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 14

set_clip_rectangle - Specify the clipping rectangle for all subsequent blt operations

SYNOPSIS

--- C ---		--- PASCAL ---
int x_left;	x-coordinate of left boundary of clip boundary	x_left: integer;
int x_right;	x-coordinate of right boundary of clip rectangle	x_right: integer;
int y_top;	y-coordinate of top boundary of clip rectangle	y_top: integer;
int y_bottom;	y-coordinate of bottom boundary of clip rectangle	y_bottom: integer;

```
set_clip_rectangle(x_left, x_right, y_top, y_bottom);
```

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under collision and clip commands).

DESCRIPTION

The clipping rectangle describes the region of the destination buffer in which a BLT operation may display pixels. The system clips (does not display) all points specified in a BLT operation which exceedthe clipping rectangle.

Clipping is performed before calling the GCP to perform a BLT to the destination buffer.

Additional detail on set_clip_rectangle may be found in the MINDSET SDG, chapter 6 (under collision and clip commands).

Refer to get_clip_rectangle (Interrupt # EF, Value of (AH) = 15H) for determining the current clip rectangle coordinates.

CAUTIONS

If clipping is disabled, then the user must ensure that the GCP does not write outside the boumds of the destination buffer.

If a clipping rectangle is specified which exceeds the size of the current destination buffer, then the clipping rectangle is truncated to the size of the current destination buffer.

Registers AX,CX,DX,DI,SI in the 80186 are altered by this routine.

REVISION

Version	Date	Comment
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 15

get_clip_rectangle - Return the current clipping rectangle coordinates

SYNOPSIS

--- C ---		--- PASCAL ---
int *x_left;	x-coordinate of left boundary of clip rectangle	**var x_left: integer;**
int *x_right;	x-coordinate of right boundary of clip rectangle	**var w_right: integer;**
int *y_top;	y-coordinate of top boundary of clip rectangle	**var y_top: integer;**
int *y_bottom;	y-coordinate of bottom boundary of clip rectangle	**var y_bottom: integer;**

get_clip_rectangle(x_left, x_right, y_top, y_bottom);

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under collision and clip commands).

DESCRIPTION

The clipping rectangle describes the region of the destination buffer in which a BLT operation may display pixels. The system clips (does not display) all points specified in a BLT operation which exceedthe clipping rectangle.

Additional detail on get_clip_rectangle may be found in the MINDSET SDG, chapter 6 (under collision and clip commands).

Refer to set_clip_rectangle (Interrupt # EF, Value of (AH) = 14H) for specifying the clip rectangle coordinates.

CAUTIONS

If clipping is disabled, then the user must ensure that the GCP does not write outside the boumds of the destination buffer.

If a clipping rectangle is specified which exceeds the size of the current destination buffer, then the clipping rectangle is truncated to the size of the current destination buffer.

Registers AX,CX,DX,DI,SI in the 80186 are altered by this routine.

REVISION

Version	Date	Comment
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH) = 16

set_collclip_detect - Enable/disable: collision/clip detection, GCP task complete interrupt. Specifies the address of collision/clip/done interrupt routine.

SYNOPSIS

--- C ---

char mode; Set conditions for interupt

Bit #	Interrupt condition
3	0 - disable, 1 - enable clipping
4	0 - disable, 1 - enable collision
6	0 - disable, 1 - enable task done interrupt

int offset; Offset of collision/clip/done interrupt service routine

int segment; Segment of collision/clip/done interrupt service routine

set_collclip_detect(mode, offset, segment);

--- PASCAL ---

mode: byte;

offset: word;

segment: word;

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under collision and clip commands).

DESCRIPTION

Disabling collision and clip detection, and interrupt on GCP task done improves system performance. The user should disable collision detection whenever collisions are not possible or not important.

Disabling clipping is highly discouraged. When clipping is disabled, it is possible for the GCP to write data outside the destination buffer, with unpredicatable results (including potential system crash).

If the value of the offset pointer is 0, then the system does not call an interrupt service routine when collision/clip/done is detected.

Additional detail on set_collclip_detect may be found in the MINDSET SDG, chapter 6 (under collision and clip commands).

Refer to get_collclip_detect (Interrupt # EF, Value of (AH) = 17H) for determining current collision/clip/done flags.

CAUTIONS

If clipping is disabled, then the user must ensure that the GCP does not write outside the boumds of the destination buffer.

Registers AX,BX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comment
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 17

get_collclip_detect - Return enable/disable: collision/clip/done detection, GCP task complete interrupt.

SYNOPSIS

--- C ---

```
char *mode;    Conditions causing interrupt
```

Bit #	Interrupt condition
3	0 - disable, 1 - enable clipping
4	0 - disable, 1 - enable collision
6	0 - disable, 1 - enable task done interrupt

```
int *offset;   Offset of collision/clip/done
               interrupt service routine
int *segment;  Segment of collision/clip/done
               interrupt service routine

    get_collclip_detect(mode, task_done, offset, segment);
```

--- PASCAL ---

```
var mode:byte;

var offset: word;

var segment: word;
```

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under collision and clip commands).

DESCRIPTION

Disabling collision and clip detection, and interrupt on GCP task done improves system performance. The user should disable collision detection whenever collisions are not possible or not important.

If the value of the offset pointer is 0, then the system does not call an interrupt service routine when collision/clip/done is detected.

Additional detail on get_collclip_detect may be found in the MINDSET SDG, chapter 6 (under collision and clip commands).

Refer to set_collclip_detect (Interrupt # EF, Value of (AH) = 16H) for specifying current collision/clip/done flags.

CAUTIONS

If clipping is disabled, then the user must ensure that the GCP does not write outside the boumds of the destination buffer.

Registers AX,BX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comment
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 18

blt_wait - Returns the collision/clip status as soon as the GCP is not busy.

SYNOPSIS

--- C ---		--- PASCAL ---
char *id;	User assigned id number for GCP operations	**var id: byte;**
int *coll_status;	Flag bit for each of last 16 blt operations 0 - no collision, 1 - collision occurred	**var coll_status: word;**
int *clip_status;8	Flag bit for each of last 16 blt operations 0 - no clip, 1 - clipping occurred	**var clip_status: word;**

blt_wait(id, coll_status, clip_status);

REFERENCE

Reference MINDSET SDG chapter 3 and 6 (under collision and clip commands).

DESCRIPTION

The blt_wait routine provides a method for obtaining the collision/clip status without the use of a user-defined collision/clip detect interrupt service routine. The system waits until the GCP is idle before returning the collision/clip status, enabling the user to synchronize program execution with the GCP if desired.

The BLT id number is the user defined reference number for the BLT operation causing one of the flags to be set. Additional information on the id number may be found in the MINDSET SDG, chapter 6, under GCP commands.

The coll_status and clip_status words contain 16 bits, one bit for each of the last 16 BLT operations (bit 0 is the first BLT, bit 15 is the more recent BLT operation). If a coll_status/clip_status bit is 0, then no collision/clip was caused by that BLT. A bit value of 1 indicates that a collision/clip was caused by that particular BLT operation, with the returned id number.

Additional detail on blt_wait may be found in the MINDSET Software Developer Guide (SDG) in chapter 6 (under collision and clip commands).

Refer to set_collclip_detect (Interrupt # EF, Value of (AH) = 16H) for specifying current collision/clip/done flags.

CAUTIONS

Registers AX,BX,CX in the 80186 are altered by this routine.

REVISION

Version	Date	Comment
0.0	07/01/84	Original version.

Interrupt # = EF
Value of (AH)= 19

NAME

blt_polygon - Draws a filled polygon into the destination buffer

SYNOPSIS

--- C ---		--- PASCAL ---
`char id;`	User assigned reference number for GCP operations	`id: byte;`
`int count;`	Number of polygon parameter sets to display	`count: integer;`
`char even_color;`	color palette index for even-numbered bits	`even_color: byte;`
`char odd_color;`	color palette index for odd-numbered bits	`odd_color: byte;`
`int xorg;`	x-origin coordinate in destination buffer	`xorg: integer;`
`int yorg;`	y-origin coordinate in destination buffer	`yorg: integer;`
`int offset;`	Offset of destination buffer	`offset: word;`
`int segment;`	Segment of destination buffer	`segment: word;`

```
blt_polygon(id, count, even_color, odd_color, xorg, yorg, offset, segment);
```

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under draw commands).

DESCRIPTION

Blt_polygon draws a filled polygon. A **parameter list** of user-defined point coordinates in memory specify the "corners" of the polygon. The blt_polygon routine automatically completes the polygon by connecting the last point in the list with the first point in the list.

Each point in the **parameter list** is defined as follows:

Byte offset	Parameter
+ 0	x coordinate in pixels
+ 2	y coordinate in pixels

The even_color and odd_color parameters are indices into the color palette and are used to create a third color by a technique known as dithering. Each pixel is evaluated for even/odd-ness by adding the (x,y) coordinates. If the sum is even, then then even_color index is used to select a color from the palette (eg. (5,5) = 10, so even_color is used). This technique is intended for use with a monitor and causes unpredictable results on a television.

If count is 1, or all points in the parameter list coincide, then blt_polygon draws a single point, using even_color/odd_color as needed.

If count is 2, or there are only two distinct points in the parameter list, then blt_polygons draws a line, using even_color/odd_color as appropriate.

Additional detail for blt_polygon may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

CAUTIONS

The fill algorithm works properly for polygons not having a boundary which crosses a horizontal line more than once. Thus, an upright hourglass will be properly filled, while an hourglass lying on its' side will not be properly filled.

Registers AX,BX,CX,DX,ES,DI,SI in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH) = 1A

blt_fellipse - Draws one or more filled ellipses into the destination buffer.

SYNOPSIS

--- C ---		--- PASCAL ---
char id;	User assigned reference number for GCP operations	id: byte;
int count;	Number of polygon parameter sets to display	count: integer;
char even_color;	color palette index for even-numbered bits	even_color: byte;
char odd_color;	color palette index for odd-numbered bits	odd_color: byte;
int xorg;	x-origin coordinate in destination buffer	xorg: integer;
int yorg;	y-origin coordinate in destination buffer	yorg: integer;
int offset;	Offset of destination buffer	offset: word;
int segment;	Segment of destination buffer	segment: word;

```
blt_fellipse(id, count, even_color, odd_color, xorg, yorg, offset, segment);
```

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under draw commands).

DESCRIPTION

The blt_fellipse routine draws a series of filled ellipses or elliptical sectors. The ellipses (or the ellipses on which the elliptical sectors are based) can be oriented either horizontally or vertically. Each ellipse is defined by a user specified parameter group

Each 16-byte **parameter group** is defined as follows:

Byte offset	Parameter
+ 0	X-center in pixels
+ 2	Y-center in pixels
+ 4	X-radius in pixels
+ 6	Y-radius in pixels
+ 8	X-begin in pixels
+ 10	Y-begin in pixels
+ 12	X-end in pixels
+ 14	Y-end in pixels
+ 16 (optional)	Pointer to next parameter group. (Refer to set_link_mode for more detail)

X-center and y-center specify the center of the ellipse in pixels from the origin. X-radius and y-radius specify the magnitude of the horizontal and vertical dimensions of the ellipse. The GCP converts negative radius values to positive values. A point is drawn by fellipses if x-radius and y-radius are 0. A line is drawn if either x-radius or y-radius is 0. X/y-begin and x/y-end specify two points through which the system draws radial vectors. An entire ellipse is drawn if the begin arc and end arc vectors are the same. An entire ellipse is also drawn if x/y-begin are 0 or if x/y-end are 0.

The even_color and odd_color parameters are indexes into the color palette and are used to create a third color by a technique known as dithering. Each pixel is evaluated for even/odd-ness by adding the (x,y) coordinates. If the sum is even, then then even_color index is used to select a color from the palette (eg. (5,5) = 10, so even_color is used). This technique is intended for use with a monitor and causes unpredictable results on a television.

If count is 1, or all points in the parameter list coincide, then blt_fellipse draws a single point, using even_color/odd_color as needed.

If count is 2, or there are only two distinct points in the parameter list, then blt_fellipse draws a line, using even_color/odd_color as appropriate.

Additional detail for blt_fellipse may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

CAUTIONS

Due to the "non-square" aspect ratio of the display screen, equating x-radius and y-radius of the ellipse does not produce a circle. For example, to draw a circle in 320x200 mode, use a y-radius/x-radius ratio of 5 to 6 (eg. y-radius = 10, x-radius = 12).

Registers AX,BX,CX,DX,ES,DI,SI in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 1B

blt_hellipse - Draws one or more hollow ellipses into the destination buffer.

SYNOPSIS

--- C ---		--- PASCAL ---
char id;	User assigned reference number for GCP operations	id: byte;
int count;	Number of polygon parameter sets to display	count: integer;
char even_color;	color palette index for even-numbered bits	even_color: byte;
char odd_color;	color palette index for odd-numbered bits	odd_color: byte;
int xorg;	x-origin coordinate in destination buffer	xorg: integer;
int yorg;	y-origin coordinate in destination buffer	yorg: integer;
int offset;	Offset of destination buffer	offset: word;
int segment;	Segment of destination buffer	segment: word;

```
blt_hellipse(id, count, even_color, odd_color, xorg, yorg, offset, segment);
```

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under draw commands).

DESCRIPTION

The blt_hellipse routine draws a series of hollow ellipses or elliptical sectors. The ellipses (or the ellipses on which the elliptical sectors are based) can be oriented either horizontally or vertically.

Refer to the description for blt_fellipse (Interrupt # EF, Value of (AH) = 1AH) for detailed definitions of the parameters.

Additional detail for blt_hellipse may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

CAUTIONS

Due to the "non-square" aspect ration of the display screen, equating x-radius and y-radius of the ellipse does not produce a circle. For example, to draw a circle in 320x200 mode, use a y-radius/x-radius ratio of 5 to 6 (eg. y-radius = 10, x-radius = 12).

Registers AX,BX,CX,DX,ES,DI,SI in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH) = 1C

save_GCP - Saves the current state of the GCP in a user-defined area of memory

SYNOPSIS

--- C ---		--- PASCAL ---
`char mode;`	0 - no save, just data_size, 1 - save GCP status	`mode: byte;`
`int offset;`	Offset of user-defined area in memory to store GCP status	`offset: word;`
`int segment;`	Segment of user-defined are in memory to store GCP status	`segment: word;`
`int data_size;`	Actual length of GCP status whether saved or not	`var data_size:integer;`
`data_size = save_GCP(mode, offset, segment);`		`save_GCP(mode, offset, segment, data_size);`

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under collision and clip commands).

DESCRIPTION

The save_GCP routine enables the user to save the current state of the GCP before the GCP is used by an interrupt service routine.

If mode has a value of 0, then save_GCP does not store the GCP status, returning only the size of the GCP data in data_size.

If mode has a value of 1, then the GCP status is stored in the memory address pointed to by offset and segment.

Refer to restore_GCP (Interrupt # EF, Value of (AH) = 1DH) for restoring the GCP status.

Additional detail for save_GCP may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

CAUTIONS

The user should employ save_GCP when using the GCP simultaneously in both normal and interrupt routines.

The user-defined area should be on the stack to ensure that all routines are re-entrant.

Registers AX,BX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH) = 1D

restore_GCP - Restores the GCP status previously stored by save_GCP.

SYNOPSIS

--- C ---		--- PASCAL ---
int offset;	Offset of user-defined area in memory where status was stored by save_GCP	**offset: word;**
int segment;	Segment of user-defined area in memory where status was stored by save_GCP	**segment: word;**
	restore_GCP(offset, segment);	

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under collision and clip commands).

DESCRIPTION

The restore_GCP routine enables the user to restore the GCP to the state it was in before GCP operations were interrupted.

Refer to save_GCP (Interrupt # EF, Value of (AH) = 1CH) for storing the GCP status.

Additional detail for restore_GCP may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

CAUTIONS

The user should employ restore_GCP when using the GCP simultaneously in both normal and interrupt routines.

The user-defined area should be on the stack to ensure that all routines are re-entrant.

Registers AX,BX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH) = 1E

fill_dest_bufer - Fills entire destination buffer with a 16-bit pattern.

SYNOPSIS

--- C ---		--- PASCAL ---
int pattern;	16-bit pattern used to fill entire destination buffer	**pattern: integer;**

fill_dest_buffer(pattern);

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under draw commands).

DESCRIPTION

Fill_dest_buffer blanks the entire destination buffer by filling the buffer word-by-word with the data in pattern. This provides a simple method of erasing, or flooding the screen.

Additional detail for fill_dest_buffer may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

CAUTIONS

The parameters specified by set_transfer_mode and set_clip_rectangle do not affect the operation of fill_dest_buffer. The fill_dest_buffer rountine always fills the entire destination buffer.

Registers AX,BX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 1F

set_font_pointer - sets the font information pointer used by the blt_string routine.

SYNOPSIS

--- C ---		--- PASCAL ---
int offset;	Offset in memory of pointer to font data	**offset: word;**
int segment;	Segment in memory of pointer to font data	**segment: word;**

set_font_pointer(offset, segment);

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under collision and clip commands).

DESCRIPTION

The set_font_pointer routine initializes a font description for use by the blt_string routine.

The data block pointed to by set_font_pointer consists of the following:

Byte offset	Parameter	Values
+ 0	font_type	0 - fixed font 1 - proportional font
+ 1	excess_white	signed byte excess inter-character white space in pixels (-128 -> 127)
+ 2	nominal_width	nominal pixel character width (not used in proportional font)
+ 4	nominal_height	nominal pixel character height (=raster height in pixels)
+ 6	bitmap_width	bitmap raster width in bytes (must be even number of bytes)
+ 8	bitmap_address	bitmap address (must be word aligned)
+ 12	first_ascii	ASCII value of first character in bitmap
+ 13	last_ascii	ASCII value of last character in bitmap

The following data is required if font_type = 1 (proportional). This data area has (last_ascii - first_ascii + 1) * 4 byte elements containing:

+ 14	bitmap_offset	offset in pixels into bitmap (-1 if character not in bitmap)
+ 16	char_width	signed byte character width in pixels (-128 -> 127)
+ 17	char_height	signed byte character height in pixels (-128 -> 127).

(char_width and char_height are used to move to next character position even if bitmap_off = -1)

Additional detail for set_font_pointer may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

Refer to blt_string (Interrupt # EF, Value of (AH) = 21H) for use of the font created by set_font_pointer.

CAUTIONS

Set_font_pointer must be called before using the blt_string routine.

Registers AX,BX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 20

get_font_pointer - Returns a pointer to the current font information.

SYNOPSIS

--- C ---		--- PASCAL ---
int *offset;	Offset of pointer to font data	**var offset: word;**
int *seg;	Segment of pointer to font data	**var segment: word;**

get_font_pointer(offset, segment);

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under collision and clip commands)

DESCRIPTION

The get_font_pointer routine returns a pointer to a data block containing current font information.

Refer to set_font_pointer (Interrupt # EF, Value of (AH) = 1FH) for details of the data structure containing font information.

Additional detail for get_font_pointer may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

Refer to blt_string (Interrupt # EF, Value of (AH) = 21H) for use of the font created by set_font_pointer.

CAUTIONS

Set_font_pointer must be called before using the blt_string routine.

Registers AX,BX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 21

blt_string - The GCP BLTs a character string using the current font into the destination buffer.

SYNOPSIS

--- C ---		--- PASCAL ---
char id;	user assigned id number for GCP operations	id: byte;
char count;	Number of character strings to blt	count: byte;
char ignore;	Number of characters to ignore at the beginning of each string blt-ed	ignore: byte;
char direction;	Direction to draw each string: 0 - left to right 1 - right to left 2 - top to bottom 3 - bottom to top	direction: byte;
char color;	Color palette index to draw strings	color: byte;
int xorg;	X-origin in destination buffer in pixels	xorg: integer;
int yorg;	Y-origin in destination buffer in pixels	yorg: integer;
int offset;	Offset in memory of string descriptor block(s)	offset: word;
int segment;	Segment in memory of string descriptor block(s)	segment: word;

```
blt_string(id, count, ignore, direction, color, xorg, yorg, offset, segment);
```

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under collision and clip commands).

DESCRIPTION

For each string, the upper left/right, and lower left corner of the first character drawn is at (x-destination + x_origin, y-destination + y_origin) specified in the **string descriptor** for the string.

The **string descriptor** is defined as follows:

Byte offset	Parameter
+ 0	x-destination in pixels (based on xorg)
+ 2	y-destination in pixels (based on yorg)
+ 4	number of characters in string
+ 6	address offset of first character
+ 8	address segment of first character
+ 10 (optional)	address offset of next string descriptor

Refer to set_link_mode (Interrupt # EF, Value of (AH) = 22H) for details on the optional parameter.

Additional detail for blt_string may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

Refer to set_font_pointer (Interrupt # EF, Value of (AH) = 1FH) for details of the data structure containing font information.

CAUTIONS

Set_font_pointer must be called before using the blt_string routine.

Registers AX,BX,ES in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 22

set_link_mode - allows the user to specify linked or contiguous mode for blt_copy, blt_copy_word, blt_string, blt_fellipse and blt_hellipse.

SYNOPSIS

--- C ---

```
char mode;     0 - contiguous parameter blocks
               1 - linked parameter blocks

                   set_link_mode(mode);
```

--- PASCAL ---

```
mode: byte;
```

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under collision and clip commands).

DESCRIPTION

Set_link_mode establishes whether parameter blocks are contiguous or linked when certain routines are called (blt_copy, blt_copy_word, blt_string, blt_fellipse, blt_hellipse).

In contiguous mode, all parameter blocks are contiguous after the first parameter block. The first parameter block is addressed from each routine with a pointer.

In linked mode, each parameter block ends with a pointer to the next parameter block in the list.

Additional detail for set_link_mode may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

CAUTIONS

Mode is set to contiguous whenever set_screen_mode (Interrupt # EF, Value of (AH) = 00H) is called.

Register AX in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 23

get_link_mode - Returns the current parameter block mode.

SYNOPSIS

--- C ---

```
char mode;     0 - contiguous parameter blocks
               1 - linked parameter blocks

mode = get_link_mode(mode);
```

--- PASCAL ---

```
var mode: byte;

get_link(mode);
```

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under collision and clip commands).

DESCRIPTION

The get_link_mode routine returns the current parameter blocks mode as specified by the most recent setlink or setmode routine call.

Refer to set_link_mode (Interrupt # EF, Value of (AH) = 22H) for details of the parameter block mode values.

Additional detail for get_link_mode may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

CAUTIONS

Mode is set to contiguous when the set_screen_mode (Interrupt # EF, Value of (AH) = 00H) routine is called.

Register AX in the 80186 is altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 24

get_GCP_status - Returns the current status of the BLTer status word

SYNOPSIS

--- C ---		--- PASCAL ---
int *gcp_idle;	GCP has completed most recent task and is idle if value is 1	**var gcp_idle: boolean;**
int *system_fly;	system is in flyback trace of video signal, GCP is inactive, but may have task pending if value is 1	**var system_fly:boolean;**
int *collision;	GCP has detected collision on last task if value is 1	**var collision: boolean;**

get_GCP_status(gcp_idle, system_fly, collision);

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under collision and clip commands).

ESCRIPTION

The get_GCP_status returns the current status of the BLTer status word. This routine is useful in a user-defined interrupt service routine to determine the status of the BLTer or cause of interrupt.

Refer to set_display_int_addr, get_diaply_int_addr, set_coll_pattern, and get_coll_pattern for additional information.

Additional detail for get_GCP_status may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

CAUTIONS

Registers AX,BX in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 25

get_char_bitmap - Returns the two addresses of the system character bitmaps.

SYNOPSIS

--- C ---		--- PASCAL ---
int *normal;	Standard font address (ASCII 0 --> 127)	**var normal: word;**
int *extras;	Extended font address (ASCII 128 --> 255) is inactive, but may have task pending	**var extras: word;**
int *segment;	Segment in memory for addresses	**var segment: word;**

get_char_bitmap(normal, extras, segment);

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under collision and clip commands).

DESCRIPTION

Two addresses are returned by get_char_bitmap; one address for the first 128 ASCII characters, the second address for ASCII characters from 128 to 255.

Refer to set_font_pointer, get_font_pointer ad blt_string for more information on fonts.

Additional detail for get_char_bitmap may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

CAUTIONS

Both addresses must be in the same memory segment.

Registers AX,ES,DI,SI in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

NAME

Interrupt # = EF
Value of (AH)= 26

get_GCP_memory - Returns memory bounds of blt-able memory.

SYNOPSIS

--- C ---		--- PASCAL ---
int *memory_flag;	Blter can address: 0 - all memory, 1 - limited memory	**var memory_flag:integer**
int *first_segment;	Segment address of first 64K bytes available to blter	**var first_segment:word;**
int *last_segment;	Segment address of last 64K bytes available to blter	**var last_segment: word;**

```
get_GCP_memory(memory_flag, first_segment, last_segment);
```

REFERENCE

Reference MINDSET SDG Chapters 3 and 6 (under collision and clip commands)

DESCRIPTION

Get_GCP_memory returns the current bounds for memory addressable by the blter. If memory_flag = 0 (all memory is addressable), then first_segment and last_segment will have a value of 0.

Additional detail for get_GCP_memory may be found in the MINDSET Software Developer Guide (SDG) in chapters 3 and 6.

CAUTIONS

Registers AX,ES,DI,SI in the 80186 are altered by this routine.

REVISION

Version	Date	Comments
0.0	07/01/84	Original version.

Section 5

Application Notes

This section contains Mindset Application Notes which may be useful during ISV development.

Application Note	Description
IDA.001	Describes the Interactive Drawing Aid (IDA) program.
COMPATIBLE.001	Documents the differences between Mindset and industry standard BIOS calls.
RS232.001	Describes the Mindset RS-232-C serial communications hardware use and BIOS calls.
SOUND.001	Describes the Sound Editor program for development of custom sound effects.

MINDSET Application Note

Interactive Design Aid (IDA) Program

App. Note # IDA.001

1 July 1984

Table of Contents

Page

Section 1

INTRODUCTION

FOREWORD

This Application Note #IDA.001 describes the Interactive Design Aid (IDA) progam version of September 9, 1983. As IDA is updated, revised application notes will be published.

As IDA is primarily intended as a Mindset development tool, current documentation is limited to a description of IDA commands. User experimentation with IDA is strongly recommended to become familiar with the program.

IDA is designed to support a variety of pointing devices, and displays a list of supported input devices when the program is run.

All user comments, suggestions or bug reports are appreciated, and should be addressed to the Mindset Certified Developer Program manager at (408) 737-8555.

CAVEATS AND GENERAL INFORMATION

IDA is written in Microsoft Pascal. Minimal trapping of I/O errors is supported in this Pascal. Often, disk errors are trapped by the DOS error handler, and you cannot return to the program. We will try to do an assembler interface for more secure error handling in the future.

File I/O is done to the disk drive you specify explicitly in the file name. If you do not specify a drive in the file name, the DOS default drive is used. Checkpoint files use the default disk drive. The disk containing IDA.EXE can be removed once the program is loaded, and replaced with a data disk if desired.

Monitor color 8 (intensified black) is displayed as black on some monitors and as dark grey on others. This is a "feature" of the particular monitor and has nothing to do with the IDA software.

All menu commands and the menu cursor are drawn in color index 15 over background color index 0. Setting color indices 0 and 15 to display the same color is highly discouraged unless the user has an exceptional memory.

FLOOD is only partially implemented. It should not flood any pixels which should not be flooded, but may miss pixels or areas which should be flooded. This will be corrected in the future.

KEYBOARD COMMANDS

Two KEYBOARD COMMANDS have been implemented:

- Hitting the space bar is equivalent to hitting the abort button on your pointing device.

- Hitting "C" or "c" suspends the current operation, allows you to move the cursor around the screen, pick a new active color by pressing the mouse button (the color to the right of the selected color will be used as the dither color), and then continues with the suspended operation. Thus you need not return to the menu to change colors.

Keystrokes are executed the next time IDA reads the pointing device position.

Keyboard commands are not operational during normal text input (such as when entering file names).

DISK FILE FORMATS

Palette files save the state of the palette. A palette file is 17 words of binary data. The first 16 words contain the palette color. The 17th word contains the index of the border color.

Raster files are variable length binary files. The first 7 words of data make up the header information:

Word 1: the version number (currently 0) of the raster file.
2: the screen mode in which the raster was created.
3: the width in bytes of the raster.
4: x size in pixels.
5: y size in pixels.
6: x offset (original x pixel location of upper left corner).
7: y offset (original y pixel location of upper left corner).

The remaining ((width in bytes)/2 * (ysize in pixels)) words of binary data represent the bitmap of the image. Each new scanline of data begins on a word boundary in the file. The first scanline of data begins with the first word following the header.

GETTING STARTED

IDA is supplied in the ISV Toolkit as an executable file, IDA.EXE. IDA may be run from either disk drive, and once loaded, the IDA program disk may be removed (See Write Rectangle for a description of disk storage requirements for creating an IDA data file).

When you run IDA it will begin by prompting for the type of device plugged into port A (9-pin connector on left side of keyboard). Enter the number associated with the device you are using. The Mindset mouse is designated as a two button ALPs mouse (number 5).

Once the pointing device has been defined, IDA will display the main menu. There are currently three menus; Main, Color and Mode. You can move from one menu to the next using the NEXT/PREVIOUS MENU commands.

The pointing device is used both to select commands, as well as to do the actual design. The use of commands on each menu is briefly described in the following three sections.

Section 2

IDA Main Menu Commands

MAIN MENU

The following commands are available from the IDA main menu, which is displayed immediately after you have selected the pointing device.

Colors

The sixteen palette colors are displayed at the top of the screen. When a color is selected, it becomes the ACTIVE color, and is used for all subsequent drawing operations, or until a new color is selected.

Each palette color is labeled 0->15. In addition, those palette entries which correspond to a valid color index in the current mode are labeled with the color index. You can not select a palette color which is not used in the current mode. See the Mindset Software Developer's Guide, Section 4, for a description of graphic modes and number of colors displayed.

Three indicators are used in the color command squares to designate the current use of the color:

B indicates that this palette index is being used as the border color.

P indicates that this palette index is being used as the primary color for graphics.

D indicates that this palette index is being used as a dither color (only used in paint and filled areas). This color is alternated with the primary color in both x and y, giving a perceived new color. Dithering has unpredicatable results on television sets, including the possiblility of 30Hz flicker in the dithered image.

A color is selected by positioning the cursor within the square and pressing the (left) button of the pointing device. The selected color will become both the primary and dither color, resulting in a "solid" (non-dithered) color for subsequent graphics.

In order to select a dither color different from the primary color, select the command DITHER COLOR and then select a color.

The border color is selected from the BORDER command on the second (color) menu.

Brushes

Below and to the left of the color palette are four "brushes" used for interactive painting. When you select a brush, the working screen will be displayed. Move the brush around with the mouse, holding down the left button when you wish to paint the screen with the active color (dithering works here). See the colors command for determining the active color.

Use the right button on the mouse to return to the menu.

NEXT MENU

Moves to the next menu screen.

This command is on all menus.

PREVIOUS MENU

Moves to the previous menu screen.

This command is on all menus.

DITHER COLOR

Allows the choice of a dither color.

If the dither color is set to be different from the active color, the dither pattern will be displayed as the background of the dither color command rectangle.

HOLLOW/FILLED

This switch determines whether CIRCLE/DRAW RECTANGLE/ELLIPSE draws a filled figure or merely the boundary. The dither color is used only for filled figures. When HOLLOW is selected, only the active color is used for drawing the lines.

NORMAL/MAGNIFIED

Allows the user to draw in a magnified mode.

This command is currently not operational.

FLOOD

Floods an area with the active color.

This command is currently only partially implemented.

In many cases, it will not flood the entire area, and may leave "holes" unfilled. This will be corrected at a later time.

The algorithm which determines the area to be flooded is:

1) Select a pixel with the cursor.

2) Every pixel of the same color as the selected pixel, and which can be reached from the selected pixel by stepping horizontally or vertically (but not diagonally) on pixels of the same color as the selected pixel, will be changed to the active color. See the colors command for determining the active color.

3) Use the right button on the mouse to return to the menu.

CHAIN LINES

Draws straight line segments using the active color.

Pushing the left button the first time defines the first point of the first line segment. Each successive push of the left button defines the end of the line segment, which then automatically becomes the beginning point of the next line segment.

Use the right button on the mouse to return to the menu.

PAIR LINES

Draws straight line segments using the active color.

Every two left button pushes defines the beginning and end of a line segment. Lines may be made joined by the left button to end and start a line segment without moving the pointing device between the two button pushes.

Lines may be made disjoint by moving the pointing device between the button push to end the previous line segment, and pressing the button to begin a newline segment.

Use the right button on the mouse to return to the menu.

CIRCLES

Every two left button pushes define the center and a point on a circle.

A circle or disk is drawn depending on the setting of the switch HOLLOW/FILLED.

Use the right button on the mouse to return to the menu.

ELLIPSES

Every two left button pushes define the center and a corner of a rectangle.

The ellipse is inscribed within the rectangle. The ellipse is filled or hollow depending on the setting of the switch HOLLOW/FILLED.

Use the right button on the mouse to return to the menu.

POINTS

Each left button push draws a single pixel in the primary color.

Use the right button on the mouse to return to the menu.

DRAW RECTANGLE

Every two left button pushes define the diagonal of a rectangle. The rectangle is filled or hollow depending on the setting of the switch HOLLOW/FILLED.

Use the right button on the mouse to return to the menu.

COPY RECTANGLE

Every two left button pushes define a rectangular region of the working screen. This rectangle is then centered on the cursor and may be dragged anywhere on the screen. A third left button push leaves the rectangle at its current position. The original rectangle is not modified.

Use the right button on the mouse to return to the menu.

MOVE RECTANGLE

Every two left button pushes define a rectangular region of the working screen. This rectangle is then centered on the cursor and may be dragged anywhere on the screen. A third left button push leaves the rectangle at its current position. The original rectangle is set to zero.

Use the right button on the mouse to return to the menu.

READ RECTANGLE

Reads a raster from disk and places it on the working screen.

The user is prompted for the file name of the raster. If the raster was not created in the same mode as the current screen mode, the command is aborted. The user is prompted to ask if the raster should be placed at its default position. The default position is the position the raster occupied when it was written to disk. If the user does not desire the default position, the raster is centered on the cursor and may be moved anywhere on the screen, being fixed in position by a left button push.

Use the right button on the mouse to return to the menu without positioning the rectangle.

WRITE RECTANGLE

Writes a rectangular region of the screen to a disk file.

The user is prompted for the disk file name. Two left button pushes define the diagonal of the rectangular region.

Note that approximately 33Kbytes of disk space are required to store the contents of the entire display screen. The user should ensure that sufficient storage space is available prior to saving the rectangle.

Use the right button to return to the menu without writing the rectangle. (A zero length file will be created if you abort during WRITE RECTANGLE).

WRITE CHECKPNT

Checkpoints the current working environment by writing the current palette into CHECKPNT.PAL and the current working screen into CHECKPNT.RAS, using the DOS default disk drive.

Section 2

IDA Color Menu Commands

The commands described in this section are available on the IDA Color Menu (Next Menu from the Main menu).

This menu allows the user to interactively modify the color palette.

BORDER

Allows the user to select the border color. Only the currently displayed border color can be modified. The current color value for the selected border is displayed below the TV/MONITOR command.

The color value can be adjusted interactively by selecting the command displaying the color component to be modified (tv or monitor RGB). Monitor color components toggle (0/1), tv color components are are incremented/decremented (0 -> 7).

MONITOR-TELEVISION

Selecting this command toggles between displaying the tv or monitor color palette.

READ PALETTE

Allows the user to read a saved palette from disk.

WRITE PALETTE

Allows the user to write the current palette to disk.

The color palette and the DITHER COLOR are found on this menu as well as the main menu.

Section 3

IDA Mode Menu Commands

These commands allow the user to select the screen mode to be used during graphics creation. Additionally, those commands which tend to destroy the current graphics state (e.g., CLEAR SCREEN), are grouped on this menu.

Select the desired screen mode from this menu.

CLEAR SCREEN

Clears (deletes) the current working screen.

All unsaved work will be lost.

This command must be selected two consecutive times to execute.

READ CHECKPNT

Reads the latest checkpoint into the current environment. All unsaved work since the latest checkpoint will be lost.

This command must be selected two consecutive times to execute.

EXIT IDA

Returns to DOS. All unsaved work will be lost.

This command must be selected two consecutive times to execute.

MINDSET Application Note

Industry Standard Compatibility Guide

Version 3.0

App. Note # COMPATIBLE.001

1 July 1984

TABLE OF CONTENTS

Section 1

Section 2

Section 3

Section 4

Section 1
FOREWORD

The purpose of this document is to provide guidelines for programmers who are developing software for the IBM P.C. and who wish to remain compatible with the MINDSET computer system. The document concentrates on those functions of the IBM P.C. which are supported by the MINDSET. Where appropriate, differences between the IBM P.C. and MINDSET are noted. The interrupt calls which are compatible on the MINDSET, particularly those related to the graphics of the IBM P.C., are documented in enough detail to allow the programmer to use the IBM BIOS without having to consult the IBM Technical Reference Manual.

When these compatibility programming guidelines are followed, software written for the IBM P.C. will run on the MINDSET and probably most of the IBM compatible machines as well. In addition, the software will be upward compatible with any new versions of the MINDSET ROM O.S. and any future architectural enhancements. Our intent is to make sure that any enhancements to this machine or future machines maintains compatibility.

The MINDSET computer offers graphics and sound capabilities that are significantly superior to the IBM P.C. and other computers in its class while offering downward software compatibility to the IBM P.C. ROM BIOS.

It is important to remember that these guidelines are meant for programmers who are developing software for the IBM P.C. It does not address or describe the unique features of the MINDSET and its ROM O.S. Specific technical details about the MINDSET are contained in the MINDSET Programmers Guide.

Section 2
PROGRAMMING GUIDELINES

2.1 OBJECTIVES

The MINDSET is designed with system characteristics which can be fully used only if certain guidelines are followed in the development of software. The characteristics are:

a. IBM PC compatibility. MINDSET is compatible with PC programs which run with the color card and which use standard ROM OS calls and/or MS-DOS 2.0 functions, or with PC-DOS 1.1 programs which map into PC-DOS 2.0.

b. Extensibility. Access to all of the special characteristics of MINDSET is provided through standard calls to the ROM O.S. in MINDSET. Future versions of MINDSET will maintain the integrity of these calls. Programs which follow the guidelines given here will be immune to changes in the MINDSET architecture and will run on MINDSET if developed for the PC by these rules.

c. The general guidelines are:

1. DO use the standard entry points provided in the IBM PC ROM BIOS.

2. DO NOT address the hardware directly.

2.2 RECOMMENDED TECHNIQUES

a. Always use the BIOS calls in the IBM PC ROM BIOS.

b. When looking for the presence of a device, access it through the ROM BIOS and check for an error code return.

c. copy protection of diskettes should be based on peculiarities of the medium.

d. Follow the Microsoft guidelines for developing BASIC programs on MS-DOS systems.

e. Writing to the PC frame buffer to speed up execution is totally compatible with the MINDSET PC modes.

f. Use a configuration program and procedure for autoload BASIC disks so that once completed, the user's program will autoload correctly.

2.3 NON-RECOMMENDED TECHNIQUES

a. DO NOT directly access the ROM BIOS data areas. This is perilous since any revision by IBM of the ROM BIOS can move the areas.

b. DO NOT sense the color card through the I/O port.

c. DO NOT use the keyboard as a controller by strobing it constantly for make/break.

d. DO NOT read directly from or write directly to the 6845 graphics processor.

e. DO NOT modify the DOS vector for 6845 parameters. This is an allowable PC DOS function which is ignored by MINDSET.

f. DO NOT require the use of a black and white monitor or address the monochrome frame buffer at B0000H.

g. DO NOT access or jump into the BASIC interpreter code.

h. DO NOT access the ROM character set directly.

i. DO NOT refer the user to specific locations or relationships for the function keys or the cursor key, e.g., on the left side of the keyboard; on the same key as the numeric keypad.

j. DO NOT use PEEKS or POKES in BASIC which access the hardware directly, access ROM BIOS data areas or which use specific ROM locations as constants.

k. DO NOT use any tie between your program logic and peculiarities of the PC, such as using specific ROM locations as constants.

2.4 INCOMPATIBILITIES BETWEEN MINDSET AND THE PC

a. MINDSET does not support the IBM PC joystick or lightpen.

b. MINDSET does not support the 160 X 100 graphics mode of the PC.

c. There is no resident ROM (cassette) BASIC in MINDSET. Autoload BASIC programs from the IBM PC will require minor changes to run on MINDSET.

d. In alpha modes, the MINDSET has 4 pages in 40 column mode; 2 pages in 80 column mode. The IBM PC has 8 pages in 40 column mode; 4 pages in 80 column mode.

2.5 COMPATIBILITY CONCEPTS

In character modes, there is a buffer of several pages of ASCII characters with attributes. There are 4 pages in 40 column modes, and 2 pages in 80 column modes. The buffer of ASCII with attributes is used during the Vblank time to draw the display buffer. Each display page has a cursor position. The cursor for the 'active page' (the page currently displayed), is drawn on the screen. The other pages' cursors are only for defining where to put or read characters. Pages are numbered starting at 0 for the first page. The IBM PC has 8 pages in 40 column modes and 4 pages in 80 column modes. We are incompatible with the IBM PC in that respect.

In IBM PC graphics modes, there is only one display buffer, and only one page. In MINDSET graphics modes where double buffering is possible, there are two pages. The first page is 0 and the second is 1. Characters may be drawn in all graphics modes. In modes where there is only 1 display page, the page parameter is ignored on the IBM PC compatible calls. The cursor position is used for character positioning. For character I/O routines in graphics modes, row means character row (8 pixel lines) and column means character column (8 pixels wide).

Rows begin at 0 for the top row on the screen, and go to 24. Columns begin at 0 for the leftmost column on the screen, and go to either 79 or 39. For write and read pixel, columns begin at 0 for the leftmost column and go to either 639 or 319, and rows begin at 0 and end at 199 for all IBM PC compatible modes. Only in MINDSET 400 scan line modes does the row number go to 399. Any illegal value for pixel row or column will be changed to the maximum allowable value.

2.6 MINDSET CHARACTER MODE DIFFERENCES

The format of the display page buffers in IBM PC character modes is an array of words. Each word contains an ASCII character code as the lower byte and an attribute as the upper byte. The attribute defines what colors the character would be drawn in for an IBM PC color monitor.

The format of the attribute is:

bit 7	blink enable
bits 6-4	red, green, blue enable for background
bits 3-0	intensify, red, green, blue enable for foreground

ATTRIBUTE	DISPLAY ON MINDSET
blink set	blink character
foreground = black background = anything	black character on white background
foreground = color background = anything	white character on black background
intensify set	no intensify

Only 2 colors are displayed, so reverse video and blink are the only attributes shown on the screen. To get other colors, the MINDSET Set Palette call may be used to redefine what color is displayed for `black' (color 0 in the frame buffer), and `white'.

2.7 MINDSET KEYBOARD DIFFERENCES

The keyboard on the MINDSET computer is laid out in the familiar typewriter keyboard format (see figure 2-1). Notice that the function keys are laid out horizontally above the alpha-numeric keys, unlike the IBM PC keyboard which locates the function keys to the left of the alpha-numeric keys. Also note the location of the keys in the cursor key section. These keys are also laid out differently on the MINDSET than on the IBM PC. Because of the differences in key locations, it is extremely important NOT to refer the application user to specific locations for keys.

Figure 2-1 Mindset Keyboard

The keyboard of the MINDSET produces scan codes and extended scan codes which are IBM PC compatible. The differences are as follows:

a. The START and PAUSE keys of the MINDSET keyboard provide additional extended scan codes 133 and 134 respectively.

b. The SYS CONFIG key of the MINDSET keyboard brings up the MINDSET System Configuration screen.

c. The RESET key of the MINDSET keyboard provides the ALT-CTRL-DEL reset function of the IBM PC keyboard.

d. There is no NUM LOCK key on the MINDSET keyboard so the cursor keys are cursor keys only and do not produce numbers.

NOTE: Because of the absence of a NUM LOCK key on the MINDSET keyboard, applications programs should not refer the user to the CTRL-NUM LOCK function of the IBM PC as a way to halt program output. Under MS DOS, the CTRL-P key will suspend screen output, and the CTRL-N key will resume it.

Section 3
MINDSET/PC INCOMPATIBILITIES

Although the Mindset computer is based on a member of the same microprocessor family as the IBM PC, not all PC programs will run on the Mindset. Due to the differences between the Intel 80186 that Mindset uses and the Intel 8088 used by the PC, as well as advances in fundamental system design in the Mindset architecture, some very basic differences exist at the hardware level. To avoid these differences in designing new programs is straightforward: use the ROM OS or MS DOS to perform I/O rather than program the hardware directly. The result of using these interfaces to the hardware is that the PC becomes effectively a subset of the Mindset from a software viewpoint. The problem is not so simple for ISVs with products on the market written for the PC.

Mindset has taken great care to be compatible with the largest possible amount of software written for the PC. However, there are some differences in the system architecture which make complete compatibility an impossibility. The next few paragraphs will describe the major areas of incompatibility from a system perspective and also list the most common types of software mismatches uncovered during our testing and verification cycles.

3.1 SYSTEM INCOMPATIBILITIES

There are several areas of incompatibilities that will cause PC software to malfunction on the Mindset. These include the interrupt structure, specific peripheral device controllers such as the keyboard and graphics interfaces, and the I/O port assignments.

3.1.1 Interrupt Structure

The interrupt controller for the Mindset is embedded in the 80186 microprocessor chip. It is not the same as the (external to the 8088) interrupt controller on the PC. Not only are there some conflicting interrupt assignments, but the method of handling them is somewhat different.

The result of these differences is that programs written for the PC that revector the device interrupts will have at least two fatal problems on the Mindset:

a. the revectored interrupt is the wrong one, and

b. the interrupt handler will not reset the interrupt properly.

3.1.2 Device Controllers

While maintaining functional, entry-point compatibility with the PC ROM OS, Mindset uses different physical hardware in performing the same or enhanced I/O. Specific differences are the DMA controller, the keyboard, and the graphics controller.

The keyboard is probably the device most frequently handled directly by applications programs other than the display. On the Mindset, a PC program will not only read the wrong port and mis-handle the interrupt, the method of using the I/O port will be incorrect.

The DMA controller for the 80186 falls into the same category as the keyboard. PC programs will not be able to properly perform DMA.

The very special case is the graphics controller. On the PC, the 6845 chip is used to control the (color monitor) video display. On the Mindset, control of the display is done through custom Mindset chips that in no way resemble the 6845. All control of the Mindset graphics is done through Mindset ROM OS calls. PC programs that make extensive use of the 6845 by addressing it directly will probably not run on the Mindset.

An example of use of the 6845 that will not work on the Mindset is setting the 160 x 200 mode (not supported on the Mindset). A further example is the use of the Video Parameter vector (1DH). Since the vectored list of parameters is specific to the 6845, Mindset ignores this vector, causing unpredictable results for PC programs.

There are some exceptions. Since there was no ROM OS function that enabled users to write to the display without screen flicker, many users were forced to address the 6845 status registers to determine when screen writes could be performed. Mindset has provided for this functionality in the ROM OS. However, since this 6845 access is so prevalent in PC programs, Mindset will emulate the status bits for those programs requiring them.

Due to hardware differences, PC programs that access joysticks or light pens cannot be used on the Mindset.

3.1.3 I/O Address Locations

Very simply put, the address space for I/O ports on the Mindset and on the PC are different. This means that the PC programs that access the I/O ports directly are looking at the wrong ports on the Mindset. In a number of cases, the device hardware registers function differently, so a direct translation of the addresses will not cure the incompatibility. However, since some devices do function similarly, Mindset has added some `translation' features that will allow the PC programs to function, although with degraded performance. See the section on I/O Emulation for more information.

3.1.4 ROM OS

Some user programs access absolute locations in the PC ROM OS. Since the Mindset ROM OS is not the same, these locations will not be the same. Additionally, Mindset does not always use the same structure to keep similar data. An example of such an incompatibility is the addressing of the graphics character font in the PC ROM OS. Not only is Mindset's font not in the same place, it is not in the same format. An additional incompatibility in the ROM OS is the number of 'pages' of text available in character modes. The Mindset is limited to only 4 pages in 40 column mode and 2 pages in 80 column mode. The PC has 8 and 4, respectively.

3.2 HIGH LEVEL INCOMPATIBILITIES

While the primary incompatibilities between the Mindset and the PC are due to hardware differences, there are some cautions at the higher levels.

3.2.1 MS DOS 2.x

Mindset uses MS DOS 2.0 as its disk operating system. Programs written for the PC that use PC DOS 1.x may experience some difficulties running under MS DOS 2.0. These difficulties are explained in the PC DOS 2.0 manual.

3.2.2 BASIC

Mindset will offer GW BASIC 2.0 with the system. While differences between this BASIC and the PC BASICA are few, it is important to note that some exist. They exist in the usual places - addressing ROM OS, DOS, or BASIC variables using absolute addresses via PEEK and POKE. Additionally, since the Mindset BASIC is packaged differently from the PC, the use of the PC software interrupts that refer to BASIC are ignored on the Mindset.

3.3 FUTURE COMPATIBILITY CONSIDERATIONS

The Mindset computer, like any other product, will be undergoing modifications and enhancements in the future. For software products to remain compatible with these changes, it is necessary for developers to follow the basic rule: USE THE BIOS. Mindset is committed to maintaining upward compatibility for products that conform to the programming practices described in the Software Developer's Guide and newsletters. The major premise of these rules is to access the features of the Mindset computer via the ROM BIOS.

A slight variation on this theme is to avoid coding of hardware addresses as constants in software products. Among the enhancements planned for the Mindset are some that may cause some of these addresses to move. Two examples are the ROM cartridge addresses and the Frame Buffer address.

Programmers developing products that run in or access the cartridge address space should not use absolute addresses for their code. This could cause several problems, among them the inability to run their programs from either cartridge slot and possible future system compatibility. Rather than coding absolute addresses, programmers should use the ROM BIOS calls to establish their execution position and set segment registers accordingly. The proper call uses INT EE, with AH = 14H. This call returns the cartridge status information from which all pertinent segment registers can be set.

Similarly, there is a ROM BIOS call that returns the address of the Frame Buffer(s). To obtain the addresses and size of the Frame Buffer(s), use INT EF, with AH = 0EH.

Using these calls will prevent two potential future product incompatibilities. Mindset's goal in providing such entry points in the ROM BIOS is to allow for extensibility in the hardware while preserving software compatibility. If you know of similar entry points that are needed to accomplish this goal, or have any suggestions for improving the utility of the ROM BIOS, please let us know as soon as you can.

Section 4

IBM/MINDSET INTERRUPT VECTORS

INTERRUPT # (IN HEX)	IBM FUNCTION	MINDSET FUNCTION
0	Divide By Zero	Same
1	Single Step	Same
2	Non-maskable Interrupt	Reserved (1)
3	Break Point Instruction	Same
4	Overflow	Same
5	Print Screen	Same
6	Reserved	Reserved (1)
7	Reserved	Reserved (1)
8	Reserved	Reserved (1)
9	Keyboard Interrupt	Reserved (1)
A	Reserved	Reserved (1)
B	Reserved	Reserved (1)
C	Reserved	Reserved (1)
D	Reserved	Reserved (1)
E	Reserved	Reserved (1)
F	Reserved	Reserved (1)
10	Video I/O Call	Same (2)
11	Equipment Check Call	Same
12	Memory Check Call	Same
13	Diskette I/O Call	Same
14	RS232 I/O Call	Same
15	Cassette I/O Call	Dummy IRET
16	Keyboard I/O Call	Same (3)
17	Printer I/O Call	Same
18	ROM Basic Entry Code	Reserved (1)
19	Boot Strap Loader	Same
1A	Time of Day Call	Same
1B	Get Control on kbd. break	Same
1C	Get Control on Timer Int.	Same
1D	Ptr. to Video Init. Table	Reserved (1)
1E	Ptr. to Diskette Parm. Tbl.	Same
1F	Ptr. to Graphics Char. Gen.	Same

(1) Reserved for Mindset ROM O.S. software. See Mindset Programmers Guide for details.

(2) Mindset supports only 2 colors in the Character Mode. Color attributes that are set for IBM software will display in black and white or in inverse video. See Interrupt #10H in this document for more information.

(3) The Mindset keyboard has no Num Lock key or numeric keypad. Cursor keys are cursor keys only and do not produce numbers.

Section 4

TABLE OF DIFFERENCES

Interrupt (hex)	Mindset Function	Differences from IBM
5	print screen	
10	video I/O	Works in Mindset video modes also. Characters in interlaced modes - 16 pixels high. Alpha mode - 4 pages in 40 columns, 2 in 80 cols (compared to IBM's 8 & 4 pages) - 2 colors only. Set mode also: sets sound mode to music. Sets screen position to configuration default. Sets palette to IBM-like palette. Disables external sync, interlaced sync. Write teletype routine really uses page # in BH, unlike IBM which requires BH to be set as active page. Get video state returns 0FFH in AL if in a Mindset video mode. Page number in BH register has meaning in Mindset double buffered graphic modes, ignored in IBM graphics modes.
11	equipment ck	
12	memory ck	
13	diskette I/O	
14	RS232 I/O	
15	Dummy IRET	Used by IBM as cassette I/O call
16	keyboard I/O	No Num Lock key or numeric keypad start=133, pause=134 scan codes (pause key is just another scan code returned, not a special trap in the keyboard routine like in the PC)

TABLE OF DIFFERENCES

Interrupt (hex)	Mindset Function	Differences from IBM
17	printer I/O	
18	Reserved	Used by IBM as the ROM BASIC entry point.
19	boot loader	Boots from a cart or disk in the order selected in the Mindset system configuration.
1A	time of day	
1B	keyboard brk	
1C	timer int.	
1D	Reserved	For IBM, this is the video parameter pointer.
1E	disk parms	
1F	alternate char. set	

DESCRIPTIONS OF IBM COMPATIBLE MINDSET INTERRUPTS AND USAGE

Interrupt Number: 05H
Function: Print Screen

Interrupt #05H is a hardware interrupt invoked whenever the PrtSc (print screen) key is depressed.

The cursor position at the time of this interrupt will be saved and restored upon return. This interrupt routine is intended to be executed with interrupts enabled. If a subsequent 'print screen' key is depressed during the time the interrupt code is printing the screen, it will be ignored. Address 50:0 contains the status of the print screen:

50:0 = 0 Either print screen has not been called or upon return from the interrupt this indicates a successful operation.

= 1 Print screen is in progress.

=377 Error encountered during printing.

==========

Interrupt Number: 10H
Function: Video I/O

This is a software interrupt entry point identical in function to the entry point supplied by IBM for video I/O. The application program performs a software interrupt 10H, with the function code passed in AH, and other parameters in other registers as specified below. All functions work in Mindset display modes as well as IBM modes, except where differences are noted. No registers are destroyed except those used to return values.

SET SCREEN MODE -- (AH) = 00H

Input Parameters:
alpha-numeric modes:
(AL) = 0 40 X 25 black and white characters
(AL) = 1 40 X 25 color characters (2 colors)
(AL) = 2 80 X 25 black and white characters
(AL) = 3 80 X 25 color characters (2 colors)
graphics modes:
(AL) = 4 320 X 200 color graphics (4 colors)
(AL) = 5 320 X 200 black and white graphics
(AL) = 6 640 X 200 black and white graphics

SET CURSOR TYPE -- (AH) = 01H

The cursor will be a rectangle 8 pixels wide during character modes and will not be displayed during graphics modes. Line 0 is the top line and line 7 is the bottom line of the character position.

Input Parameters:
(CH) = Bits 3-0 = start line for cursor (may be 0 - 7)
(CL) = Bits 3-0 = end line for cursor (may be 0 - 15)

If either CH or CL is an illegal value, or if CH is > CL, no cursor will be displayed. If CL is > 7 and < 16 the end line parameter will be changed to 7.

SET CURSOR POSITION -- (AH) = 02H

Input Parameters:
(DH) = character row number
(DL) = character column number
(BH) = Page number (must be 0 for IBM graphics modes)

If in graphics mode and if there is only one page, BH is ignored. In character modes, page is masked to a legal value. In all modes, if DL > maximum column, column = maximum column. In all modes, if DH > maximum row (24), then row = 24.

READ CURSOR POSITION -- (AH) = 03H

Input Parameters:
(BH) = Page number (must be 0 for graphics modes)
Output Parameters:
(DH) = character row number
(DL) = character column number
(CH) = start line of cursor (bits 4 - 0 only)
(CL) = end line of cursor (bits 4 - 0 only)

In graphics modes if there is only one page, BH is ignored. In character modes, page is masked to a legal value.

SELECT ACTIVE DISPLAY PAGE -- (AH) = 05H

(valid for alpha modes only)
This sets which ASCII/attribute buffer will be displayed on the screen in character modes, but only sets which graphics page will be used for scroll active page in Mindset graphics mode. It does not change which page is displayed in Mindset graphics modes.

Input Parameters:
(AL) = New page value (0-3 for modes 0 & 1)
(0-1 for modes 2 & 3)
No effect in graphics mode if there is only one page. Page is masked to a legal value.

SCROLL ACTIVE PAGE UP -- (AH) = 06H

Input Parameters:
(AL) = number of lines to scroll.
(AL) = 0 means fill with fill attribute byte
(CH,CL) = row, column of upper left corner of scroll
(DH,DL) = row, column of lower right corner of scroll
(BH) = attribute (alpha modes) or color (graphics modes) to be used on blank line

Fill byte in graphics modes for lines scrolled out of. The area in the rectangle from upper left to lower right corner inclusive will be scrolled up by the number of lines in AL.

In character modes, the lines scrolled out of will be filled with spaces, with the attribute specified in BH. In graphics modes, the lines will be filled with the color byte passed in BH.

If AL is 0, or if AL is more than the number of lines in the rectangle, the entire area will be filled with the fill pattern.

SCROLL ACTIVE PAGE DOWN -- (AH) = 07H

Input Parameters:
(AL) = number of lines to scroll. Lines blanked at top of window.
(AL) = 0 means blank entire window
(CH,CL) = row, column of upper left corner of scroll
(DH,DL) = row, column of lower right corner of scroll
(BH) = attribute (alpha modes) or color (graphics modes) to be used on blank line

READ ATTRIBUTE/CHAR AT CURSOR POSITION -- (AH) = 08H

In character modes, this reads the character and attribute from the ASCII/attribute buffer.
In graphics modes, this compares what is in the graphics buffer to the stored bit maps of what a character looks like in 8 x 8 pixel representation. The character may be any combination of foreground colors on a background of background color (0) pixels to be recognized. The first character that matches what was found in the graphics buffer is the one returned in AL. The characters are searched starting from character 0.
In graphics modes, no attribute is returned.

Input Parameters:
(BH) = display page (valid for alpha modes only)
Output Parameters:
(AL) = character in ASCII (0 if not found in graphics modes)
(AH) = attribute for character modes, garbage for graphics modes.

WRITE ATTRIBUTE/CHAR AT CURSOR POSITION -- (AH) = 09H

In character modes, this puts the character(s) and attribute(s) into the ASCII/attribute buffer.
In graphics modes, BL is the color. If bit 7 of BL is set, the character will be XOR'ed into the frame buffer.
In graphics modes, this draws the character(s) from an 8 x 8 pixel representation stored in ROM, or from the alternate character set 8 x 8 pixel representation supplied by the application, if the character is > 127.
In 400 pixel line graphics modes, the character is drawn as 8 pixels wide by 16 pixels high so that it looks the same as the characters in the other modes.

Input Parameters:
(BH) = display page (valid for alpha modes only)
(CX) = count of characters to write
(AL) = character to write
(BL) = attribute of character (alpha/color of character (graphics)
see WRITE DOT for bit 7 of (BL) = 1

WRITE CHARACTER ONLY AT CURSOR POSITION -- (AH) = 0AH

Input Parameters:
(BH) = display page (valid for alpha modes only)
(CX) = count of characters to write
(AL) = character to write
(BL) = color (graphics modes)

SET COLOR PALLETTE -- (AH) = 0BH

This chooses colors for IBM display modes only.

Input Parameters:
(BH) = pallette color ID
0 for background and border color in IBM graphics modes
1 for foreground colors in IBM graphics modes
0 for border color in character modes
(BL) = color value for background or foreground as chosen in BH
If BH = 0:
BL = color choice 0 through 15 - IRGB value
Character Modes:
BL = border color
IBM 320 x 200 graphics mode:
BL = background and border color
IBM 640 x 200 graphics mode:
BL = foreground color
If BH = 1:
IBM 320 x 200 graphics mode only:
BL = foreground color set choice
0 = red/green/yellow
1 = cyan/magenta/white

This interrupt is not as flexible as the Mindset Set Palette call, but it works (in IBM modes only) for compatibility.

WRITE DOT -- (AH) = 0CH

Input Parameters:
(DX) = pixel row
(CX) = pixel column
(AL) = color value:
if bit 7 of (AL) = 1, color value is XOR'ed into the frame buffer.
This call does not work in character modes.

READ DOT -- (AH) = 0DH

Input Parameters:
(DX) = pixel row
(CX) = pixel column
Output Parameters:
(AL) = color of pixel
This call does not work in character modes.

WRITE TELETYPE -- (AH) = 0EH

Write character using backspace, return, linefeed, and bell on display page passed in BH.

Input Parameters:
(AL) = ASCII character to write
(BL) = foreground color (graphics modes)
(BH) = display page in alpha mode

This call just checks for the special characters (bell, return, linefeed, backspace), calls write_char_only, then updates the cursor position.
If the cursor position was at the rightmost column on the screen, the character (and the cursor position) goes to the next line.
A carriage return goes to column 0 of the current line. A linefeed goes to the next line, current column, and scrolls the entire display page up to do that if necessary.
A bell causes a beep noise from the sound module and the beeper, if the beeper on the Mindset is enabled.
A backspace moves back one column (no erase of the character) unless it is already at column 0. If the character is a linefeed, and the cursor was already on the last line of the page, the page is scrolled up one line and the new line is filled with blanks.
On the IBM PC, this call only works for active page. On the Mindset, the call works for any valid page passed in BH.

CURRENT VIDEO STATE -- (AH) = 0FH

Output Parameters:
(AL) = mode (as in IBM_set_mode)
(AH) = number of columns of characters on the screen
(BH) = current active display page

If the current mode is not an IBM compatible mode, 0FFH is returned in the AL register.

==========

Interrupt Number: 11H
Function: Equipment Check

When interrupt #11H is called, the system attempts to determine what optional peripherals are attached to the system.

INPUT:

(none required)

OUTPUT:

(AX) = bits set which indicate the following:

BIT 15,14	= number of printers attached
BIT 13	= not used
BIT 12	= game I/O attached
BIT 11,10,9	= number of RS232 cards attached
BIT 8	= not used
BIT 7,6	= number of disk drives attached 00=1 01=2 10=3 11=4 (only if bit 0 = 1)
BIT 5,4	= initial video mode 00 = not used 01 = 40X25 b/w using color card 10 = 80X25 b/w using color card 11 = 80X25 b/w using b/w card
BIT 3,2	= Planar RAM size
BIT 1	= not used
BIT 0	= this bit indicates that there are disk drives on the system

==========

Interrupt Number: 12H
Function: Memory Check

When interrupt #12H is called the system determines the amount of memory in the system.

INPUT:

(none required)

OUTPUT:

(AX) = number of contiguous 1K blocks of memory in system

Interrupt Number: 13H
Function: Disk I/O

Interrupt #13H provides an interface with the disk drive system.

INPUT:

(AH) = 0 reset the disk system
(AH) = 1 read the status of the system into (AL)
the disk status from the last operation is used
Registers for READ/WRITE/VERIFY/FORMAT:
(DL) = drive number (0-1)
(DH) = head number (0-1)
(CH) = track number (0-39)
(CL) = sector number (1-9 in DOS 2.0)
(AL) = number of sectors (maximum = 9)

(ES:BX) = address of buffer (not required for verify)

(AH) = 2 read desired sectors into memory
(AH) = 3 write desired sectors from memory
(AH) = 4 verify desired sectors
(AH) = 5 format the desired track
When using the FORMAT operation, the buffer pointer (ES,BX) must point to the collection of address fields for the track. Each field must be 4 bytes, (C,H,R,N), where:
C = track number
H = head number
R = sector number
N = number of bytes per sector:
00 = 128 bytes
01 = 256 bytes
02 = 512 bytes
03 = 1024 bytes
There must be one entry for every sector on the track.

OUTPUT:

(AH) = status of operation
80H = no response from device
40H = seek operation failed
20H = NEC controller failed
10H = bad CRC when reading disk
09H = attempted DMA across 64K (cannot occur on Mindset)
08H = DMA overrun on operation
04H = Record not found
03H = attempted write on write protected disk
02H = address mark not found
01H = bad command passed to disk I/O
(CY) = 0 successful operation
(CY) = 1 failed operation
For READ/WRITE/VERIFY
DS,BX,DX,CH,CL preserved
AL = number of sectors read

==========

Interrupt Number: 14H
Function: RS232 I/O

Interrupt 14H provides a byte stream I/O to the communications port. The following parameters are required:

INPUT:

(AH) = 0 Initialize the communications port
(AL) Contains the parameters for initialization according to the following bit patterns:

7	6	5	4	3	2	1	0
-- BAUD RATE --			-PARITY-		STOPBIT	WORD LENGTH	
000 = 110			X0 = NONE		0 = 1	10 = 7 BITS	
001 = 150			01 = ODD		1 = 2	11 = 8 BITS	
010 = 300			11 = EVEN				
011 = 600							
100 = 1200							
101 = 2400							
110 = 4800							
111 = 9600							

On return, conditions are set as in the call to communications status (AH=3)--see below.

(AH) = 1 Send the character in (AL) over the comms line (AL) is preserved
On exit: Bit 7 of AH is set if the routine was unable to transmit the byte of date over the line. The remainder of AH is set as in a status request, reflecting the current status of the line.

(AH) = 2 Receive a character in (AL) from the comms line before returning to the caller.
On exit: AH has the current line status, as set by the status routine, except that the only bits left on are the error bits (7,4,3,2,1). In this case the timeout bit indicates data set ready was not received. Therefore, AH is non-zero only when an error has occurred.

(AH) = 3 Return the comms port status in (AX)
AH contains the line control status:
bit 7 = timeout
bit 6 = transmission shift register empty
bit 5 = transmission holding register empty
bit 4 = break detect
bit 3 = framing error
bit 2 = parity error
bit 1 = overrun error
bit 0 = data ready
AL contains the modem status:
bit 7 = received line signal detect
bit 6 = ring indicator
bit 5 = data set ready
bit 4 = clear to send
bit 3 = delta receive line signal detect
bit 2 = trailing edge ring detector
bit 1 = delta data set ready
bit 0 = delta clear to send

(DX) = parameter indicating which RS232 card (0,2 allowed)

OUTPUT:

AX modified according to parameters of call
all others unchanged

==========

Interrupt Number: 16H
Function: Keyboard I/O

Interrupt 16H provides an interface to the keyboard.

INPUT:

(AH) = 0 read the next ASCII character struck from the keyboard and return the result in (AL), scan code in (AH)

(AH) = 1 set the Z flag to indicate if an ASCII character is available to be read.
(ZF) = 1 no code available
(ZF) = 0 code is available
If ZF = 0 then the next character in the buffer to be read is in AX, and the entry remains in the buffer.

(AH) = 2 Return the current shift status in AL register
80H = insert state is active
40H = caps/lock state has been depressed
10H = scroll lock state has been toggled
08H = alternate shift key depressed
04H = control shift key depressed
02H = left shift key depressed
01H = right shift key depressed

OUTPUT:

As noted above, only AX and flags are changed
All registers retained

==========

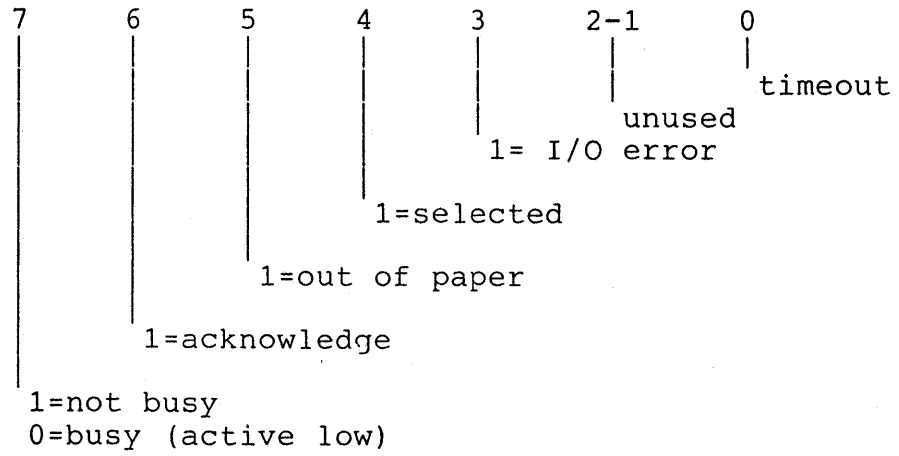
Interrupt Number: 17H
Function: Printer I/O

Interrupt #17H provides communication with a printer.

(AH) = 0 Print the character in (AL)
On return, AH=1 if the character could not be printed (timeout). Other bits are set as on normal status call.

(AH) = 1 Initialize the printer port
Returns with (AH) set with the printer status

(AH) = 2 Read the printer status into (AH)



(DX) = Printer to be used (0,1,2) which correspond to the actual values in PRINTER_BASE area.

OUTPUT: AH is modified, all others unchanged

==========

Interrupt Number: 19H
Function: Boot Strap Loader

Control is passed to the highest priority program medium available.

==========

Interrupt Number: 1AH
Function: Time of Day

Interrupt #1AH allows the 24-hour clock to be set or read.

INPUT:

(AH) = 0 Read the current clock setting
Returns:
CX = high portion of count
DX = low portion of count
AL = 0 if the timer has not passed 24 hours since the last read.
<> 0 if on another day

(AH) = 1 Set the current clock
CX = high portion of count
DX = low portion of count

Note: Counts occur at the rate of 1193180/65536 counts/second or about 18.2 per second.

==========

Interrupt Number: 1FH
Function: Pointer to graphics character generator

This is the space where the application may store a pointer to its own bit map of characters for drawing characters from 128 through 255. In all graphics modes, this bit map must be stored as eight by eight bit arrays for each character, even if the mode is more than one bit per pixel. If this bit map is not supplied, the character bit maps stored in Mindset format in the ROM O.S. will be used to draw the characters. In character modes, this extra set of characters is ignored. The read character calls also use the application supplied character set, if any, to try to match whatever bit pattern is in the display buffer. The IBM PC will display garbage for any character between 128 and 255 in graphics mode if this character set pointer is not supplied. The Mindset computer will display from its own character set for the extra characters if there is no pointer supplied.

==========

MINDSET Application Note

RS-232C Data Communications

App. Note # RS232.001

July 1, 1984

Mindset Corp.
617 N. Mary Ave.
Sunnyvale, CA. 94086

Table of Contents

Section 1

INTRODUCTION

Introduction

This document describes the use of data communications with the Mindset computer. It is written for software developers who are familiar with the development of data communications software for the IBM PC.

In general, the data communications hardware section of the Mindset computer is not totally compatible with that of the IBM PC. One major difference is in the interrupt structure used by Serial I/O devices. Whereas the IBM PC has a dedicated interrupt vector for Serial I/O devices, the Mindset computer shares a single interrupt among several types of I/O devices. A second major difference is in the I/O port addresses used to access the serial chip in the module.

Mindset RS-232 Communications

All data communications I/O in the Mindset computer are performed by Mindset I/O modules. An I/O module is an encased circuit board that slides into a slot in the back of the Mindset computer. There are six such slots in a fully configured Mindset. Each slot has a unique "base address" through which the module that is plugged into that slot can be addressed. Each module contains a code by which the software can identify the type of module that is plugged into each slot. There are two Mindset I/O modules that can be used for data communications, the RS-232-C module and the 300 baud modem module.

RS-232-C Module

This is a general purpose, serial I/O device which has an RS-232 connector for input and output. This connector has the same pinout description as the IBM Asynchronous Adapter, but has a female connector on it rather than the male connector used by IBM. The serial I/O chip that is used in this module is the National Semiconductor INS8250 UART, the same one that is used in the IBM Asynchronous Adapter card, but with with different register I/O addresses.

This module can be used for communicating with an external modem or for communicating directly with a terminal or another computer. The module supports all standard transmission rates from 110 baud to 9600 baud.

300-Baud Modem Module

This is a double-width communications module that is a direct-connect modem compatible with a Bell 103 modem. It contains no RS-232 connector, only an RJ-11 jack that can be used to connect the modem to any telephone line that has a modular plug. The module also uses the National Semiconductor INS8250 UART chip for its serial I/O and supports transmission rates of 110 and 300 baud.

In addition to the modem function, the 300-baud Modem Module supports autodialing. Unlike many autodialing modems, which are controlled by sending serial data, the Mindset 300-baud Modem Module is controlled through parallel registers on the circuit board that are independent of the serial I/O function. The chip used to control all of the autodialling functions is the TMS 99531 Dialer which provides for both pulse and tone dialing.

Section 2

IBM PC Compatible Data Communications

IBM PC Compatible BIOS Calls

The Mindset computer contains a ROM-based program that emulates all of the IBM PC's BIOS calls, including Interrupt 14H, the RS-232-C I/O Interrupt. This interrupt provides functions to set the communications baud rate, set the state of the RS-232 output signals, read the status of the RS-232 input signals and the status of the serial I/O chip, and to read and write serial data.

It is assumed that the reader of this document is familiar with this interrupt and the use of these functions.

The Mindset BIOS software implements all of the features (both good ones and bad ones) of the equivalent IBM BIOS software. For example, the software prevents data from being sent with a "send data" call until both "Clear To Send" and "Data Set Ready" signals have been asserted. Also, in both systems, these calls do not provide "interrupt-driven" I/O. It is therefore possible, and in many cases likely, that high-speed, incoming data will be lost if the user's program is not able to check for input data often enough.

Another BIOS call that is useful in conjunction with the RS-232-C I/O call is Interrupt 11, Equipment Check. This allows the program that is running to determine how many serial devices are installed. Note that the BIOS software makes this determination at the time of system power-on, or when the system is reset with the ALT-RESET key combination. For this reason as well as for the safety of the hardware, I/O modules should not be inserted into nor removed from the system while the power is turned on.

The autodialing functions of the 300-baud modem are not accessible through these IBM-compatible BIOS calls, since these functions are not invoked through the serial data stream.

If more than one accessible module is installed in the Mindset computer, each module will be assigned an index based on the I/O slot into which it is installed. This index is used as a parameter to all communications calls and determines which module is being addressed. Modules that are installed in the base unit will be assigned a lower index than those installed in the expansion unit (containing disk drive(s)). Within each each unit, a module installed nearer the left side (when viewed from the back) will have a lower index than one installed to the right of it. For example, a module installed in the left-most slot of the base unit will always be assigned index 0. Also, if there is only one communications module plugged in, it will always be assigned index 0, regardless of which slot it is installed in.

IBM PC Compatible Basic Commands

Mindset's GW-BASIC supports all of the communications functions that are included in IBM's BASIC and BASICA. These functions include "OPEN COM" (and all of its options), "CLOSE", "LOC" to determine how many input characters are available, and "EOF". Formatted and unformatted I/O is supported using "INPUT", "PRINT", "READ", and "WRITE". Refer to Mindset's GW-BASIC Reference Manual for descriptions of these commands.

In general, the Mindset implementation of GW-BASIC communications functions will have superior performance to the equivalent functions running in an IBM system. This means that communications programs can run at a faster data rate without losing data.

Mindset Hardware Emulation

The Mindset computer includes some special features that allow it to map direct, machine language references to the IBM Asynchronous Adapter into equivalent functions on Mindset Serial I/O modules. References to I/O port port addresses 03F8H through 03FFH will be mapped to the Mindset module corresponding to BIOS communications device index 0, which is MS-DOS device "COM1". References to I/O port addresses 02F8H through 02FFH will be mapped to the Mindset module corresponding to BIOS comm device index 1 which is MS-DOS device "COM2".

This technique is provided only to allow certain programs to run without modification and is not recommended for new development. Because of the methods used to map from one system to another, the IBM format for all data should be followed exactly regardless of other information contained in this document with the one exception (noted below). The technique has the following known differences and restrictions:

Bit 3 of Port 03FCH (or 02FCH) is inverted from the IBM Asychronous Adapter. This is the INS8250 Modem Control Register.

	Enable Interrupts
Mindset	Bit 3 = 0
IBM	Bit 3 = 1

There is no emulation for replacing the Communications Interrupt Vector. Setting an address into the IBM vector will not result in the transfer of control to the user's interrupt routine and will most likely cause the Mindset processor to "hang", requiring the power to be cycled off, then on again to clear this condition.

I/O port accesses to these addresses can not be made in an interrupt routine.

In summary, this technique cannot be used in an interrupt environment. Furthermore, most communications programs that do not require interrupts can be handled using the IBM-compatible BIOS calls on the Mindset computer. The use of BIOS calls is preferable to using the Hardware Emulation.

Section 3

Mindset-Native Data Communications

Mindset-Unique BIOS Calls

Mindset has included in its extended set of BIOS calls several calls that are used to perform data communications I/O. These calls are used in conjunction with the IBM-compatible BIOS calls to initialize and maintain a communications line. All of these extended BIOS calls are documented in Section 9 of the Mindset Software Developer's Guide. A developer using these features should have this guide, which is included as part of the Programmer's Development Library (PDL). The remainder of this section gives some supplementary information for each of the commands.

The Mindset-unique data communications calls can be divided into four categories:

Modem Status

RS-232-C Get Modem Status - Use this command only if you wish to determine the type of Mindset hardware that is associated with a particular index. Many applications will simply be configured by the user to use COM1 or COM2 and the application can use index 0 or index 1 without regard to the type of device that it corresponds to. The information returned about the 1200-baud module is incorrect and should not be used; it is provided only for future expansion. The IBM-compatible "Equipment Check" call can be used to determine the total number of serial devices, including the RS-232-C modules. The index returned by Get Modem Status can be used with both Mindset-unique and industry-compatible communications commands.

Modem Initialization

RS-232-C Set Input Buffer - This command must be executed before any data can be received. Be sure that the buffer size passed is the word size not the byte size. The actual number of data elements that can be stored in the input buffer is one less than the size of the buffer; a buffer size of one should never be specified. This command can also be used to reinitialize the buffer to flush any unwanted data.

RS-232-C Set Output Buffer - This command must be executed before any data can be sent. The buffer size specified for this command is the byte size since only one byte per data element is required for output. The actual number of data elements that can be stored in the output buffer is one less than the size of the buffer; a buffer size of one should never be specified. This command can also be used to reinitialize the buffer to flush any unwanted data.

Modem Control

RS-232-C Get Buffer Status - In a real-time program, this command should always be issued before an attempt is made to read or write data to determine whether or not input data is available or sufficient output space is available. Note that both BX and CX are used to return information; the user should save and restore these registers if the program is using them. As described above, the total number of elements in the buffer is one less than the size allocated. Therefore, if an output buffer were configured with a buffer size of 16, and no data had been sent to that buffer, a Get Buffer Status call would return a value of 15 empty bytes.

RS-232-C Set Communications Control - Bits 2-7 of AL should always be set to 0 to avoid undesirable side effects; bits 2-7 of BL are "don't care" bits. This command must be executed with bits 0 and/or 1 of BL set to 1 before any reception and/or transmission of data can take place. Setting these bits enables the interrupts, but it does not require the user's program to handle these interrupts. It merely allows the data to be written to or read from the buffers by the BIOS itself when an interrupt occurs. These interrupt control bits can also be used to start and stop the flow of data. For example, if a program receives an indication that a remote device is not ready for more data, it can disable the transmit interrupt, preventing any data that is in the output buffer from being sent. When the remote device is again ready to receive data, the transmit interrupt be should be set and the flow of data will pick up where it left off. If it is no longer desirable to send that buffered data, it can be eliminated by reinitializing the buffer with the RS-232-C Set Output Buffer Command. A similar technique may be used to temporararily disable data reception.

Data Transfer

RS-232-C Get Communications Control - Bits 2-7 of both AL and BL are indeterminate.

RS-232-C Send Character - This command only writes data into the buffer; in order to have the data actually sent, the user must make sure that the transmit interrupt is enabled using the RS-232-C Set Communications Control command. Unlike the industry-compatible Send Data command, this command never waits for any modem status signals, such as Clear to Send (CTS) or Data Set Ready (DSR) before sending the data. If the user requires these functions, he should check the status of these lines using the industry-compatible call before issuing the Send Character call.

RS-232-C Get Character - This command retrieves data and status information from the buffer when available. Use the Get Buffer Status command to determine whether or not data is available in order to avoid having the command wait continuously for input data. The receive interrupt must be enabled using the Set Communications Control command before any data can be received into the buffer.

RS-232-C Send Character String - This command works just like the Send Character command. It simply allows the user to transmit a sequence of characters using a single call.

RS-232-C Set Communications Break - This command allows the user to set and clear a communications break condition. The duration of the break is controlled by the user.

Interrupt Handling

Using the Mindset-unique BIOS calls allows most communications programs to be written without any user interrupt handling. The user merely instructs the Mindset BIOS routines to handle all communications interrupts and to operate on the user-specified input and output buffers.

There is a method provided by the Mindset BIOS for transferring program control to a user routine when an interrupt occurs. This is useful for programs that do not want the overhead of regularly scanning the input buffer when no data is available, programs that follow an existing structure that uses I/O interrupts, and certain other applications in which interrupts are important.

The Mindset BIOS call that is used to invoke this function is called "Set Module Interrupt" and is documented in Section 10 of the Mindset Software Developer's Guide (SDG). This call can be used under the following guidelines:

1) The Module Interrupt is not device-specific. This means that program control may be passed to the user's routine because of some other type of interrupt. The user's interrupt handling routine must not assume that the device that it handles actually generated an interrupt and should simply do a return if there is no activity on its device.

2) When using "direct I/O", that is, reading and writing the Mindset module I/O addresses directly, incoming serial data will not be available from the 8250 chip. The Mindset BIOS handles the interrupt and reads the input data from the 8250 before the BIOS passes control to the User Module. Therefore, direct I/O may not be used within a module interrupt service routine.

3) Hardware emulation I/O (i.e., using IBM compatible I/O addresses and letting the system map them to Mindset addresses) may not be used within a module interrupt service routine.

4) The user's program need not issue any type of "End of Interrupt" (EOI) command to the interrupt controller. The Mindset BIOS handles that completely. The user should not ever attempt to access the IBM-compatible interrupt controller (8259 chip).

MINDSET Application Note

Stereo Sound Editor Program

App. Note # SOUND.001

1 July 1984

Table of Contents

Section 5
Sound List Formats

Section 1

INTRODUCTION

FOREWORD

This Application Note #SOUND.001 describes the Mindset Sound Editor program (V1.2) dated November 17, 1983. As the Sound Editor is updated and revised, new application notes will be published.

STEREO SOUND EDITOR USER'S GUIDE

The Mindset Stereo Sound Editor enables the user to generate sounds by setting sound module parameters experimentally. The sound editor provides the capability to generate single sounds or lists of sounds to be played back in sequence. Sound lists may be saved in a disk file for later retrieval; they can also be edited until the desired sound sequence has been composed.

The sound module is capable of playing up to six different voices simultaneously, on each of two channels. Three separate sound modes are provided; each mode enables the activation of a specific subset of parameters. The modes of the two channel need not be the same since there are two separate sound modules. The sound modes are as follows:

Mode 1: Four musical voices with limited special effects
Mode 2: Two voices with special music and noise effects
Mode 3: Six voices with no special effects

The sound editor knows which special effects parameters are active based on the selected mode and voice. Each sound consists, as a minimum, of a frequency and an amplitude. Optional parameters depend upon user selections.

Section 2

Parameters of a Sound

PARAMETERS OF A SOUND

The parameters of a sound are as follows:

Frequency
The frequency range of a sound is 0 to 4999 Hertz.

Amplitude
The amplitude, or volume, control varies from 0 to 255.

Attack/Decay
The attack/decay controls are enabled via function key 1; selecting this option automatically deselects AM modulation. The attack/decay control are used to increase/decrease the volume of a sound automatically. Every 3.2 milliseconds, the sound processor adds the attack value to the amplitude value until the amplitude value becomes greater than 255; at this time, attack is disabled and decay is enabled. The decay value is then subtracted from the amplitude until the next subtraction would result in a negative value. At this time, decay is disabled.

AM Modulation
The AM modulation control is enabled via function key 2; selecting this option automatically deselects attack/decay. This parameter is used to vary the volume of a sound over time; the modulation frequency increases from 0 to 127, and decreases again for values from 128 to 255.

FM Modulation
The FM modulation control is enabled via function key 3; selecting this option automatically deselects the FM frequency ramping option. FM frequency causes the lower byte of the sound frequency to vary from zero to 255 at a rate proportional to the modulation frequency.

FM Frequency Ramping
The FM frequency ramping control is enabled via function key 4; selecting this option automatically deselects the FM modulation option. The FM frequency ramp is a 16-bit value which is added to the frequency every 3.2 milliseconds, until the addition result exceeds 7FFF (hex) in the frequency counter; the frequency is then decremented until it reaches zero.

Noise Mask

Mode 2 of the sound processor includes a noise mask which contains a random 8-bit number. The noise mask is ANDed with this value to produce a varying degree of noise; this value is then exclusive ORed with the high byte of the frequency counter to produce a distorted sound.

Sine Mask

Mode 2 of the sound processor includes a sine table address mask register to enable the specification of various waveforms. The sine table contains 256 entries in the following positions:

```
entry   0  =  sine 0     radians
entry  64  =  sine pi/2  radians
entry 128  =  sine pi    radians
entry 192  =  sine 3pi/2 radians
```

Section 3

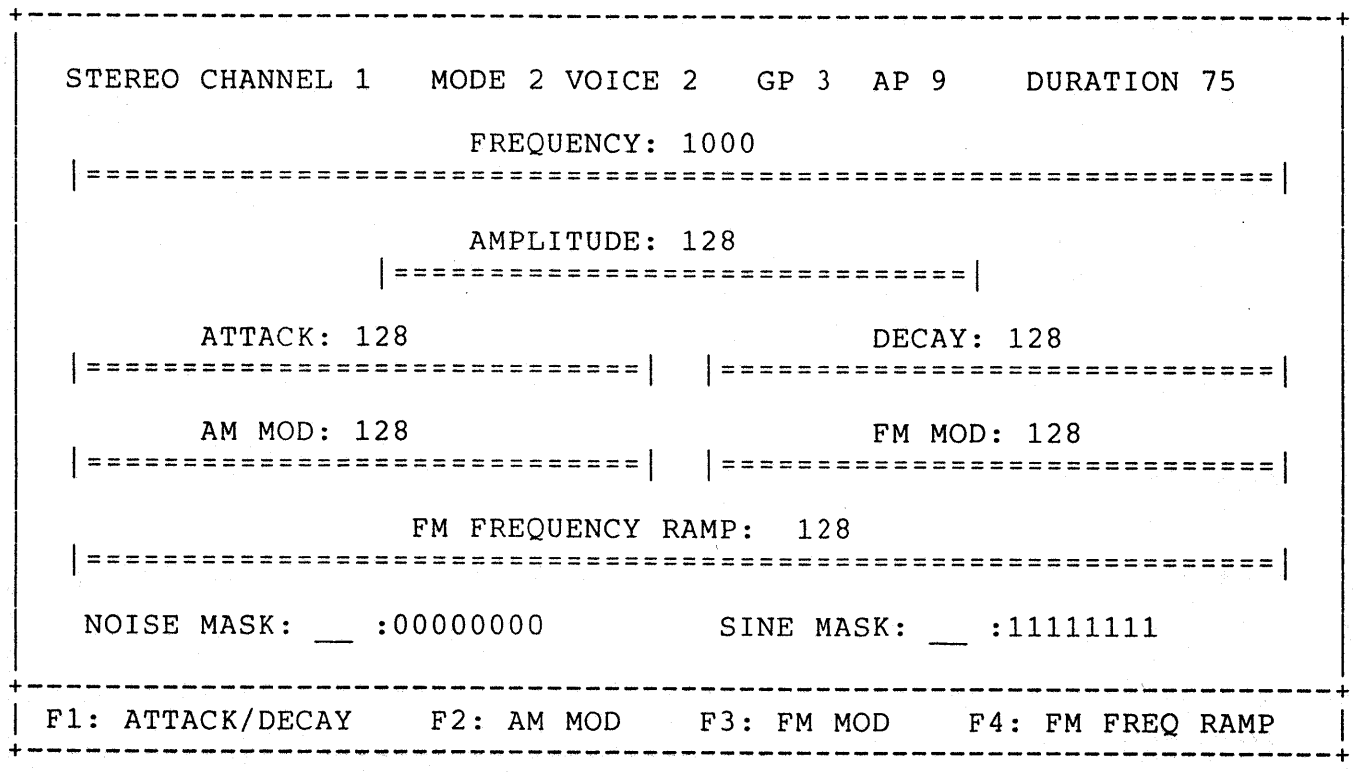
Display Format

DISPLAY FORMAT

The sound editor display has been designed to present a visual image of the sound parameters. Except for the noise mask and sine mask, each parameter is represented by a scale; a scale cursor shows the parameter's current value. This value is also displayed in digital format.

The frequency parameter is incremented non-linearly; low frequency increments along the scale are smaller than high frequency increments.

The parameter currently being specified is shown in a highlighted window.



Note: Display shows the positions of the parameter scales; only a subset of these scales actually appears on the screen at any time.

Section 4

Sound Editor Commands

SOUND EDITOR COMMANDS OVERVIEW

All commands refer to the active voice, displayed on the status line.

The status line lists the following editor parameters:

(1) Stereo or monophonic mode
(2) Active channel (irrelevant if stereo mode selected)
(3) Sound mode
(4) Active voice
(5) Get pointer (GP) in sound list position of the previous 'get'
(6) Append pointer (AP); current length of soundlist
(7) Sound duration in multiples of 1/60 second

The highlighted parameter may be modified in one of three ways:

(1) Digital input:
Enter decimal values terminated by <cr>, <=>, <.>, </>, <up-cursor>, or <down-cursor>.
The parameter is then assigned the value entered or the maximum allowable value, whichever is greater.

(2) Left or right cursor:
The parameter value is changed by an parameter-dependent increment.

(3) Control-left or control-right cursor:
The parameter value is changed by 16 times the increment value.

The active parameter is selected by using the up-cursor and down-cursor keys, and is shown in a highlighted window.

Mode 2 includes the sine and noise mask parameters which are specified and displayed in binary. After 8 digits have been entered, the program prevents further inputs for that parameter.

Function keys F1 thru F10 are used to select and deselect sound options and play the sounds and soundlists, as follows:

F1	Select Attack/Decay option
F2	Select AM Modulation option
F3	Select FM modulation option
F4	Select FM frequency ramp option
F5	Select channel (toggle)
F6	Select stereo or monophonic mode (toggle)
F7	Not used
F8	Play all soundlists simultaneously, based on F5/F6
F9	Play entire soundlist for active voice, based on F5/F6
F10	Play sound currently displayed on screen

Hitting F8, F9, or F10 while a sound or soundlist is playing causes the sound editor to start the newly requested action.

SOUND EDITOR COMMANDS

The sound editor is commanded by entering single letter commands. All commands requiring parameters will prompt for such parameters. Prompted inputs must be terminated by <return>.

The commands are as follows:

Append sound specified by current parameter settings to the end of the soundlist. You might want to listen to the sound (via F10) before appending it.

Begin a loop of sounds in soundlist. The iteration feature enables the user to repeat sound sequences by enclosing them in 'begin - end' loop pseudo statements. Nesting of these loops is not permitted. Note: The editor does not check the syntactical correctness of the 'begin - end' loops. A 'begin' without an 'end' may be hazardous to the sound editor's health.

Clear soundlist of active voice (displayed in status line).

Display soundlist for active voice. The user may temporarily stop the output of the list via the control-s key, and resume the output via the control-q key. After the entire list has been displayed, the user's next keyboard entry is used to refresh the screen; that key is not treated as a command entry.

Erase a sound in the soundlist; program prompts for list position to be erased.

Get a sound from soundlist and display it on screen. It becomes the active sound; previous contents are lost.

Help; displays this list of available commands.

Insert sound specified by current parameter settings into soundlist; program prompts for insert position.

Load a soundlist from disk; program prompts for filename. Before executing this command, select the desired mode and voice; the soundlists are stored on disk without voice or mode identification and are loaded into the soundlist buffer of the currently selected active voice.

Set sound **M**ode to value entered (1,2,3 are legal). Voice set to 1.

ENd a loop of sounds in the soundlist; program prompts for number of loop iterations.
Note: The editor does not check the syntactical correctness of the 'begin - end' loops. An 'end' without a 'begin' is hazardous to the editor's health.

Put (replace) sound specified by current parameter settings into soundlist; program prompts for soundlist position.

Quit program; turns off vblank interrupts.

Save soundlist on disk; program prompts for filename.

Set sound dura**T**ion; user specifies duration in multiples of 1/60 second.

Set acti**V**e voice; program prompts for voice (1-6 legal, depending on mode).

Write a music file, where a music file is defined as Voices 1-4 of the 1st and 2nd channels (Voices 5-6 are not processed). To write a music file, set up your soundlists as before. When all lists are ready to be saved, enter **W**. This command is similar to the Save and Load commands, except that it works on all soudlists for modes 1 and 2.

EXecute a music file (defined in Write, above). This command reads a previously written music file back into the soundlist. This command is similar to the Save and Load commands, except that it works on all soundlists for modes 1 and 2.

Error messages related to the commands are displayed on line 25 of the displays.

At any time, **<escape>** causes the display to refresh, removing any command and error messages on lines 24 and 25.

Section 5

Sound List Formats

SOUND LIST FORMAT DESCRIPTION

Sound lists are stored in a format designed to minimize record size; the records are processed directly by the sound processor which converts them to the 26-byte sound data required by the sound chip. The final record of a soundlist contains all zeros; this record is automatically appended by the editor.

The sound list does NOT contain the mode or the voice parameters; therefore, a sound list constructed and saved in one mode and/or voice may be loaded and played in another mode and/or voice, with unpredictable results.

A sound record consists of 6 words, as follows:

LOW BYTE	HIGH BYTE	Word
Duration (1 to 255)	Amplitude (0 to 255)	1
Frequency (0 to 5000 Hz), equiv. to 0-32767 (note 2)		2
Option control (note 1)	00	3
Attack (0 to 255)	Decay (0 to 255)	4 (a)
00	AM modulation (0 to 255)	4 (b)
00	FM modulation (0 to 255)	5 (a)
FM frequency ramp (0 to 65535)		5 (b)
Sine mask (00H - FFH)	Noise mask (00H - FFH)	6

Note 1: option control bits specify the selected options and are as follows:

Bit 3: attack/decay
Bit 2: FM frequency ramp
Bit 1: FM modulation
Bit 0: AM modulation

Note 2: frequency is converted in two ways, based on sound mode.
For modes 1 and 3: factor = 6.5536
For mode 2: factor = 8.0282

Frequency in hertz is then computed by: value/factor.

Appendix A

Useful Tables

This appendix contains several useful tables of data. These tables contain data frequently referenced during program development.

Documentation Guide -- Groups reference documents by those topics associated with high-level language program development.

Hardware Configuration -- Describes the minimum and recommended Mindset computer hardware configurations required for program development.

Program Development Environment -- Lists the directories of the Program Development Environment and the Program Development diskettes when used with the ISV Toolkit.

Toolkit Diskette Configuration -- Lists the directories of the three diskettes provided with the ISV Toolkit; C and PASCAL language library diskettes, and a diskette containing two Mindset utility programs, IDA and Sound Editor.

Graphical Shapes to Library Routines -- This table is intended to serve as a guide for selecting a workable combination of library routines to create a given type of shape. Shapes are grouped as point, polygonal and ellipsoidal.

ASCII Character Set and Keyboard Scan Codes -- This list provides a handy cross-reference of ASCII characters, decimal and hex values, and corresponding Mindset keyboard scan code.

Numerical List of Library Routines -- This list is ordered by the interrupt number used by the library routine, and sub-ordered by the function value in register AH. This list contains all possible interrupt/functions on the Mindset, even if there is no library function. For each routine that does exist, the library routine name is listed.

Alphabetical List of Library Routines -- This list is ordered alphabetically by library routine name. For cross-reference, the interrupt/function values are also listed. This list contains only currently implemented library routines.

Graphical Shapes to Library Routines -- This table is intended to serve as a guide for selecting a workable combination of library routines to create a given type of shape. Shapes are grouped as point, polygonal and ellipsoidal.

ASCII Character Set and Keyboard Scan Codes -- This list provides a handy cross-reference of ASCII characters, decimal and hex values, and the corresponding Mindset keyboard scan code.

ISV Toolkit Documentation Guide

Topic	Reference Documents	ISV Toolkit
Assembly Lanaguage	INTEL iAPX-86/88 User's Manual INTEL iAPX-186/188 User's Manual PDL Volume, Macro Assembler Manual PDL Volume, MS-DOS Programmer's Reference Manual	User's Guide, Section 2
Linking	Introductory Guide to MS-DOS	User's Guide, Section 3
C Language	'The C Programming Language', Kernighan and Ritchie Microsoft/Lattice C Language Reference Manual	User's Guide, Section 4 & 5, Appendix B
PASCAL Language	'PASCAL User Manual and Report', Jensen and Wirth Microsoft PASCAL Language Reference Manual	User's Guide, Section 4 & 5, Appendix C
MINDSET Graphics	PDL Volume, Software Developer's Guide	User's Guide, Appendix A
MS-DOS Interface	PDL Volume, MS-DOS Programmers Reference Manual	User's Guide, Section 5
Interactive Drawing Aid (IDA)	Mindset Application Note #IDA-001	User's Guide, Section 6
Sound Editor	Mindset Application Note #SOUND-001	User's Guide, Section 5 & 6
RS-232 Data Communications	Mindset Application Note #RS232-001	User's Guide, Section 5 & 6
IBM/Mindset Compatibility Guide	Mindset Application Note #COMPAT-001 PDL Volume, Software Developers Guide	User's Guide, Section 6
Example Programs	'The C Programming Language', Kernighan and Ritchie 'PASCAL User's Manual and Report', Jensen and Wirth	User's Guide, Appendix B & C
ISV Toolkit Language	Library	User's Guide, Section 5
Mindset System	PDL Volume, Software Developer Guide, Appendix D	

ISV Toolkit
Hardware Configuration

The minimum, and recommended Mindset computer hardware configurations are listed in the following table.

Configuration	Base Unit	Expansion Unit 1-Drive	Expansion Unit 2-Drive	Options	Notes
Minimum	Required	Required		None	1
Recommended	Required		Required	Printer Module	2,3

Notes:

1) This minimum configuration requires the additional 96K bytes of RAM in the Expansion Unit, for a total system memory of 128K bytes, with one 360K byte disk.

2) With only one disk drive, the Program Development Environment diskette contents must include all files with a Note 1, from BOTH the Program Development Environment and Program Development diskette contents (see Appendix A).

3) This is the configuration for which the ISV Toolkit is designed. Total system memory is 256K bytes, with two 360K byte disk drives. The printer module is optional, but is required for connecting a parallel printer.

C Language Program Development Environment

Diskette Directories

The C Language Program Development Environment diskette contains all files required to edit, link and compile a C program. This diskette should be placed in disk drive A: (left drive) after booting the Mindset computer with the MS-DOS V2.xx operating system. The ISV Toolkit contains batch execution files which expect this disk in drive A:. On a single disk drive system this diskette may also contain the C program source, to eliminate the need to exchange the Program Development Environment diskette and the Program Development diskette.

Diskette	Filename		Size	Source	Notes
Program Development Environment	(should contain MS-DOS V2.00 or higher)				
	lc1	...	64384	User	1,2
	lc2	...	65792	User	1,2
	editor	...	27638	User	1,2
	link	...	42340	User	1,2,3
	lcs	lib	79360		1,2
	cs	obj	1003		1,2
	MSBIOSEE	C	3487		4
	MSBIOSEF	C	14976		4
	DOS	C	2299		4
	IBMBIOS	C	2821		4
	BIOS	LIB	10752		1
	C	BAT	19		1
	CUSER	INC	4625		1
	CALLBIOS	ASM	3801		4
	BLTCHAR	ASM	4480		4
	VBLANK	ASM	768		4

NOTES:

1) Files required as a minimum for program development.
2) User must copy these files from purchased software.
3) LINK.EXE is distributed with the Mindset Program Developer Library.
4) Files are optional on Program Development Environment diskette.

C Language Program Development Environment

Diskette Directories

The C Language Program Development diskette contains the user C program source, and all files required during compilation and linking. This diskette should be placed in disk drive B: (right hand drive) after the Mindset computer has been booted with MS-DOS V2.xx. The ISV Toolkit contains batch execution files which expect this diskette in drive B:. On single disk drive systems, the user C program source, and those files with Note 1 (see below) may be transferred to the C Language Program Development Environment diskette. The resulting single diskette should be placed in disk drive A: (left hand drive).

Diskette	Filename		Size	Source	Notes
Program Development	(should not contain MS-DOS)				
	lcs	lib	79360	User	1,2
	BIOS	LIB	10752		1
	CUSER	INC	4626		1
	cs	obj	1003	User	1,2
	EVB	OBJ	290		4
	SETBLOCK	OBJ	152		4
	BLTCHAR	OBJ	333		4
	GETMEM	OBJ	352		4
	CC	BAT	73		1
	DOEXAMPL	BAT	256		4
	POINT	C	768		4
	COLOR0	C	2816		4
	COLOR0	EXE	10912		4
	COLOR1	C	3328		4
	DC	C	3840		4
	DRAW	H	512		4
	DRAW	C	3968		4
	CURSOR	C	2304		4
	IDA_FRUT		32014		4
	FRUIT	C	3968		4
	TEXT	C	10752		4
	FONT1		2294		4

NOTES:

1) Files required as a minimum for program development.
2) User must copy these files from purchased software.
3) LINK.EXE is distributed with the Mindset Program Developer Library.
4) Files are optional on Program Development diskette.

PASCAL Language Program Development Environment

Diskette Directories

The PASCAL Language Program Development Environment disk contains all files required to edit, link and compile PASCAL programs. This disk should be placed in disk drive A: (left drive) after booting the Mindset computer with the MS-DOS V2.xx operating system. The ISV Toolkit contains batch execution files which expect this disk in drive A:. On a single disk drive system this diskette may also contain the PASCAL program source, to eliminate the need to exchange the Program Development Environment diskette and the Program Development diskette.

Diskette	Filename		Size	Source	Notes
Program Development Environment	(should contain MS-DOS V2.00 or higher)				
	pas1	...	93696	User	1,2
	pas2	...	93568	User	1,2
	pas3	...	22144	User	1,2
	link	...	42330	User	1,3
	editor	...	14336	User	1,2
	BIOS	CST	8832		1
	BIOS	PAS	38656		4
	BIOS	EXT	14720		1
	BIOS	TYP	1684		1
	ASM	PRF	313		4
	PASIFC	ASM	7523		4
	COPYRT	PAS	1063		4
	COPYRT	ASM	1042		4

NOTES:

1) Files required as a minimum for program development.
2) User must copy these files from purchased software.
3) LINK.EXE is distributed with the Mindset Program Developer Library.
4) Files are optional on Program Development Environment diskette.

PASCAL Language Program Development Environment

Diskette Directories

The PASCAL Language Program Development diskette contains the user's program source, and all files required during compilation and linking. This diskette should be placed in disk drive B: (right hand drive) after the Mindset computer has been booted with MS-DOS V2.xx. The ISV Toolkit contains batch execution files which expect this diskette in drive B:. On single disk drive systems, the PASCAL program source, and those files with Note 1 (see below) may be transferred to the PASCAL Language Program Development Environment diskette. The resulting single diskette should be placed in disk drive A: (left hand drive).

Diskette	Filename		Size	Source	Notes
Program Development	(should not contain MS-DOS)				
	BIOS	OBJ	11675		1
	PASIFC	OBJ	474		1
	pascal	lib	102912		1,2
	nulf	obj	640		1,2
	nule6	obj	640		1,2
	fink		3300		1,2
	paskey		2861		1,2
	pasuxu	obj	768		1,2
	filuxu	obj	5504		1,2
	conuxu	obj	1024		1,2
	BIOS	TYP	1684		1
	BIOS	EXT	14720		1
	PASCAL	BAT	88		1
	POINT	PAS	2048		4
	COLOR0	PAS	5632		4
	COLOR0	EXE			4
	COLOR1	PAS	5760		4

NOTES:

1) Files required as a minimum for program development.
2) User must copy these files from purchased software.
3) LINK.EXE is distributed with the Mindset Program Developer Library.
4) Files are optional on Program Development diskette.

ISV Toolkit

Diskette Directories

Diskette Label	Filename		Size
ISV Toolkit - 1 of 3			
	MSBIOSEE	C	3487
	MSBIOSEF	C	14976
	DOS	C	2299
	IBMBIOS	C	2821
	BIOS	LIB	10752
	C	BAT	19
	CUSER	INC	4625
	COPYRT	PAS	1063
	COPYRT	ASM	1042
	CALLBIOS	ASM	3801
	BLTCHAR	ASM	4480
	BLTCHAR	OBJ	333
	VBLANK	ASM	768
	EVB	OBJ	290
	SETBLOCK	OBJ	152
	GETMEM	OBJ	352
	CC	BAT	73
	DOEXAMPL	BAT	256
	POINT	C	768
	COLOR0	C	2816
	COLOR0	EXE	10912
	COLOR1	C	3328
	DC	C	3840
	DRAW	H	512
	DRAW	C	3968
	CURSOR	C	2304
	IDA_FRUT		32014
	FRUIT	C	3968
	TEXT	C	10752
	FONT1		2294
ISV Toolkit - 2 of 3			
	BIOS	CST	8832
	BIOS	PAS	38656
	BIOS	EXT	14720
	BIOS	TYP	1684
	BIOS	OBJ	11675
	ASM	PRF	313
	PASIFC	ASM	7523
	PASIFC	OBJ	474
	COPYRT	PAS	1063
	COPYRT	ASM	1042
	PASCAL	BAT	88
	POINT	PAS	2048
	COLOR0	PAS	5632
	COLOR0	EXE	
	COLOR1	PAS	5760

ISV Toolkit

Diskette Directories

Diskette Label	Filename		Size
ISV Toolkit - 3 of 3			
	IDA	EXE	94450
	SOUND	EXE	80510

Cross Reference of Graphical Shapes

to Library Routines

Graphical Shape	Library Routine(s)	Comments
POINT		
1 to n-points	blt_polypoint	1) Each call to blt_polypoint sets pallete color to use.
POLYGONAL		
1-Line n-Lines	blt_polyline	1) Draws line segments, or connected. 2) Palette color may be set for each line drawn.
Triangle Square/ Rectangle n-gon	blt_polygon	1) Fills the polygon with selected palette color. 2) Two palette colors may be dithered for the fill color. 3) Degenerate case include; only one point, and two points to draw a line.
ELLIPSOIDAL		
Arc Circle Ellipse	blt_ellipses	1) Ellipses may be filled with a color or hollow. 2) Adjusting parameters draws arcs, circles and ellipses.

------- ASCII -------			-MINDSET-
Dec	Hex	Character	Scan Code
0	0	NUL (^@)	29,75,03
1	1	SOH (^A)	29,30
2	2	STX (^B)	29,48
3	3	ETX (^C)	29,46
4	4	EOT (^D)	29,32
5	5	ENQ (^E)	29,18
6	6	ACK (^F)	29,33
7	7	BEL (^G)	29,34
8	8	BS (^H)	14
9	9	HT (^I)	15
10	A	LF (^J)	29,36
11	B	VT (^K)	29,37
12	C	FF (^L)	29,38
13	D	CR (^M)	28
14	E	SO (^N)	58
15	F	SI (^O)	58
16	10	DLE (^P)	29,25
17	11	DC1 (^Q)	29,16
18	12	DC2 (^R)	29,19
19	13	DC3 (^S)	29,31
20	14	DC4 (^T)	29,20
21	15	NAK (^U)	29,22
22	16	SYN (^V)	29,47
23	17	ETB (^W)	29,17
24	18	CAN (^X)	29,45
25	19	EM (^Y)	29,21
26	1A	SUB (^Z)	29,44
27	1B	ESC (^[)	29,26
28	1C	FS (^\)	29,43
29	1D	GS (^])	29,27
30	1E	RS (^^)	29,54,07
31	1F	US (^_)	29,54,12
32	20	SPACE	57
33	21	!	42,02
34	22	"	42,41
35	23	#	42,04
36	24	$	42,05
37	25	%	42,06
38	26	&	42,08
39	27	'	41
40	28	(	42,10
41	29	)	42,11
42	2A	*	42,09
43	2B	+	42,13
44	2C	`	40
45	2D	-	12
46	2E	.	52
47	2F	/	53
48	30	0	11
49	31	1	02
50	32	2	03

------- ASCII -------			-MINDSET-
Dec	Hex	Character	Scan Co
51	33	3	04
52	34	4	05
53	35	5	06
54	36	6	07
55	37	7	08
56	38	8	09
57	39	9	10
58	3A	:	54,39
59	3B	;	39
60	3C	<	54,51
61	3D	=	13
62	3E	>	54,52
63	3F	?	54,53
64	40	@	42,03
65	41	A	30
66	42	B	48
67	43	C	46
68	44	D	32
69	45	E	18
70	46	F	33
71	47	G	34
72	48	H	35
73	49	I	23
74	4A	J	36
75	4B	K	37
76	4C	L	38
77	4D	M	50
78	4E	N	49
79	4F	O	24
80	50	P	25
81	51	Q	16
82	52	R	19
83	53	S	31
84	54	T	20
85	55	U	22
86	56	V	47
87	57	W	17
88	58	X	45
89	59	Y	21
90	5A	Z	44
91	5B	[	26
92	5C	\	43
93	5D	]	27
94	5E	^	54,07
95	5F	<-	n/a
96	60	.	52
97	61	a	30
98	62	b	48
99	63	c	46
100	64	d	32
101	65	e	18

ASCII			-MINDSET-
Dec	Hex	Character	Scan Code
102	66	f	33
103	67	g	34
104	68	h	35
105	69	i	23
106	6A	j	36
107	6B	k	37
108	6C	l	38
109	6D	m	50
110	6E	n	51
111	6F	o	24
112	70	p	25
113	71	q	16
114	72	r	19
115	73	s	31
116	74	t	20
117	75	u	22
118	76	v	47
119	77	w	17
120	78	x	45
121	79	y	21
122	7A	z	44
123	7B		
124	7C	|	42,43
125	7D		
126	7E	~	42,40
127	7F	ESC	01

Character/Key	-MINDSET- Scan Code
CTRL	29
LEFT SHIFT	42
RIGHT SHIFT	54
ALT	56
CAPS LOCK	58
F1	59
F2	60
F3	61
F4	62
F5	63
F6	64
F7	65
F8	66
F9	67
F10	68
SCR LCK	70
HOME	71
CURSOR UP	72
PAGE UP	73
CURSOR LEFT	75
CURSOR RIGHT	77
END	79
CURSOR DOWN	80
PAGE DOWN	81
INSERT	82
DEL	83
START	84
PAUSE	85
SYS CONFIG	86
RESET	87
BREAK	88

INT ## (AH) DEC/HX	Function	ISV Toolkit Library Routine
	--- IBM ---	
INT 10	**IBM compatible graphics**	
00/00	Set video mode (from AL)	set_ibm_mode
01/01	Set cursor start and end pixel line	
02/02	Set cursor pos. on page	set_cursor_position
03/03	Get " " " "	get_cursor_position
04/04	Read light pen	
05/05	Set active display page	
06/06	Scroll page section up	
07/07	" " " down	
08/08	Read char./attribute at cursor	
09/09	Write " " " "	write_char_only
10/0A	Write character only at cursor	write_char_cursor
11/0B	Select colors for IBM display modes only	
12/0C	Write one pixel in graphics	write_dot
13/0D	Read " " " "	read_dot
14/0E	Write teletype character	write_teletype
15/0F	Return current video mode	
INT 11	**Equipment Check**	
/	Return power-on equip. list	
INT 12	**Memory Size**	
/	Return memory switches set	

INT ## (AH) DEC/HX	Function	ISV Toolkit Library Routine
	--- IBM ---	
INT 13	**Disk I/O**	
00/00	Reset diskette system	
01/01	Get status of diskette(s)	
02/02	Read specified sectors	
03/03	Write " "	
04/04	Verify " "	
05/05	Format specified track	
INT 14	**RS-232C I/O**	(see MINDSET INT EE fncs. 40-49)
INT 16	**Keyboard I/O**	
00/00	Get next ASCII char.	get_kb_char
01/01	Check for char. avail.	test_kb_buffer
02/02	Get keyboard status	get_kb_shift_status
INT 17	**Printer I/O**	
00/00	Print character in AL	
01/01	Initialize printer port	
02/02	Get printer port status	

INT ## (AH) DEC/HX	--- DOS --- Function	ISV Toolkit Library Routine
INT 20	**Program terminate**	
INT 21	**MS-DOS function calls**	
00/00	Program terminate	
01/01	Wait for keyboard input	
02/02	Write character to display	
03/03	Wait for auxiliary input	
04/04	Write character to AUX	
05/05	Write character to printer	
06/06	Direct console I/O	
07/07	Direct console input(no echo)	
08/08	Wait for ketboard input (no echo)	
09/09	Print string	
10/0A	Buffered keyboard input	
11/0B	Get standard input status	
12/0C	Clear standard input buffer	
13/0D	Reset disk file buffers	
14/0E	Select disk drive	
15/0F	Open disk file	
16/10	Close " "	
17/11	Search for first entry	
18/12	Search " next "	
19/13	Delete disk file	
20/14	Sequential disk block read	
21/15	Sequential " " write	
22/16	Create disk file	
23/17	Rename " "	

INT ## (AH) DEC/HX	--- DOS --- Function	ISV Toolkit Library Routine
INT 21	**MS-DOS function calls**	
24/18	DOS Internal Use Only	
25/19	Get current disk drive code	
26/1A	Set disk transfer address	
27/1B	Get allocation table info. (for default disk drive)	
28/1C	Get " " " (for specified disk drive)	
29/1D to 32/20	DOS Internal Use Only	
33/21	Random disk block read	
34/22	Random " " write	
35/23	Disk file size	
36/24	Set random record field	
37/25	Set interrupt vector	
38/26	Create new program segment	
39/27	Random disk block read	
40/28	Random " " write	
41/29	Parse filename	
42/2A	Get date from MS-DOS	
43/2B	Set " " "	
44/2C	Get time " "	
45/2D	Set " " "	
46/2E	Set/reset verify switch	
47/2F	Get disk transfer address	
48/30	Set DOS version number	
49/31	Terminate but remain resident	

INT ## (AH) DEC/HX	--- DOS --- Function	ISV Toolkit Library Routine
INT 21	**MS-DOS function calls**	
50/32	DOS Internal Use Only	
51/33	Ctrl-Break check	
52/34	DOS Internal Use Only	
53/35	Get interrupt vector	
54/36	Get disk free space	
55/37	DOS Internal Use Only	
56/38	Get country dependent info.	
57/39	Create disk sub-directory	
58/3A	Delete " "	
59/3B	Change " "	
60/3C	Create disk file	dos_file_create
61/3D	Open " "	dos_file_open
62/3E	Close " " handle	dos_file_close
63/3F	Read disk file by "	dos_file_read
64/40	Write " " " "	dos_file_write
65/41	Delete disk file	dos_file_delete
66/42	Move file read/write pointer	dos_file_lseek
67/43	Change disk file mode	
68/44	I/O control for devices	
69/45	Duplicate disk file handle	
70/46	Force duplicate disk file handle	

INT ## (AH) DEC/HX	--- DOS --- Function	ISV Toolkit Library Routine
INT 21	**MS-DOS function calls**	
71/47	Get current disk directory	
72/48	Allocate memory	
73/49	Free allocated memory	
74/4A	SETBLOCK (modify memory blocks)	
75/4B	Load/execute disk program	
76/4C	Terminate process and exit	
77/4D	Get return code of sub-process	
78/4E	Find first matching filename	
79/4F	Find next " "	
80/50-83/53	DOS Internal Use Only	
84/54	Get verify state	
85/55	DOS Internal Use Only	
86/56	Rename a disk file	
87/57	Get/set disk file date/time	

INT ## (AH) DEC/HX	Function	ISV Toolkit Library Routine
	--- DOS ---	
INT 22	Terminate address	
INT 23	Ctrl-Break address	
INT 24	Critical error handler vector	
INT 25	Absolute disk read	
INT 26	Absolute disk write	
INT 27	Terminate but stay resident	
INT 28-3F	DOS Reserved	

INT ## (AH) DEC/HX	Function	ISV Toolkit Library Routine
	--- UNIQUE ---	
INT EE	**MINDSET unique commands**	
00/00	Write TTY string	
01/01	Write TTY string w/attributes	
02/02	Set display device	set_display_device
03/03	Set screen position	
04/04	Get " "	
05/05	Set cursor shape	
06/06	Set display sync features	set_sync_mode
07/07	Set display interrupt	set_display_int
08/08	Set mode of real time clock	
09/09	Get " " " " "	
10/0A	Set real time clock status	
11/0B	Get " " " "	
12/0C	Get time from real time clock	
13/0D	Set time on " " "	
14/0E	Get date from real time clock	
15/0F	Set date on " " "	
16/10	Get alarm from real time clock	
17/11	Set alarm on " " "	
18/12	Set real time clock interrupt	
19/13	Reserved	
20/14	Read ROM/RAM cartridge status	
21/15	Format RAM cartridge	
22/16	Get RAM cartridge dir. info	
23/17	Put " " " "	
24/18	Read RAM cartridge file	
25/19	Write " " "	
26/1A	Delete blocks from RAM cartridge file	

INT ## (AH) DEC/HX	--- UNIQUE --- Function	ISV Toolkit Library Routine
INT EE	**MINDSET unique commands**	
27/1B	Reserved	
28/1C	Reserved	
29/1D	Reserved	
30/1E	Reserved	
31/1F	Joystick control input	joystick
32/20	Get module ID table	
33/21	Turn off system power	
34/22	Set system LED's	set_led
35/23	Print string	
36/24	Set sound mode	sound_mode
37/25	Set sound register	sound_regs
38/26	Sound data	sound_data
39/27	Stereo module check	stereo_check
40/28	RS-232C Send character	
41/29	" Get "	
42/2A	" Send string	
43/2B	" Get "	
44/2C	" Set input buffer	
45/2D	" Set output "	
46/2E	" Set comm. control	
47/2F	" Get " "	
48/30	" Get modem status	
t9/31	" Set BREAK	
50/32	Set Aux out	
51/33	Set user module interrupt	
52/34	Set system timer rate	
53/35	Get " " "	
54/36	Enable/disable beeper	enable_beeper
54/37	Test beeper ON	test_beeper
57/38	Set beeper ON/OFF	set_beeper

INT ## (AH) DEC/HX	Function	ISV Toolkit Library Routine
	--- GRAPHICS ---	
INT EF	**MINDSET extended graphics**	
00/00	Set screen mode	set_screen_mode
01/01	Get " "	get_screen_mode
02/02	Set transfer mode	set_transfer_mode
03/03	Get " "	get_transfer_mode
04/04	Set destination buffer	set_dest_buffer
05/05	Get " "	get_dest_buffer
06/06	Set write mask	set_write_mask
07/07	Get " "	get_write_mask
08/08	BLT copy	blt_copy
09/09	BLT copy word	blt_copy_word
10/0A	Set palette	set_palette
11/0b	Get "	get_palette
12/0C	BLT polypoint	blt_polypoint
13/0D	BLT polyline	blt_polyline
14/0E	Get buffer information	get_buffer_info
15/0F	Set display interrupt address	set_display_int_addr
16/10	Get " " "	get_display_int_addr
17/11	Switch active buffer	switch_active_buffer
18/12	Set collision pattern	set_collision_pattern
19/13	Get " "	get_collision_pattern
20/14	Set clip rectangle	set_clip_rectangle
21/15	Get " "	get_clip_rectangle
22/16	Set collision/clip detect	set_collclip_detect
23/17	Get " "	get_collclip_detect

INT ## (AH) DEC/HX	--- GRAPHICS --- Function	ISV Toolkit Library Routine
INT EF	**MINDSET extended graphics**	
24/18	GCP wait	blt_wait
25/19	BLT polygon	blt_polygon
26/1A	BLT filled ellipses	blt_ellipse
27/1B	BLT hollow "	blt_hellipse
28/1C	Save GCP	save_GCP
29/1D	Restore GCP	restore_GCP
30/1E	Fill destination buffer	fill_dest_buffer
31/1F	Set font pointer	set_font_pointer
32/20	Get " "	get_font_pointer
33/21	BLT string	blt_string
34/22	Set parameter block mode	set_link_mode
35/23	Get " " "	get_link_mode
36/24	Get GCP status	get_GCP_status
37/25	Get character bitmap address	get_char_bitmap
38/26	Get GCP memory bounds	get_GCP_memory

Appendix B

Example C Programs

These C programs are provided as examples for using the ISV Toolkit. The program listing is shown for each program, which includes comments. The compile, link and go batch file used is described in Section 4, Examples for C Library.

POINT.C	- Draws a single point on screen
COLOR0.C	- Displays framed 16 color palette on screen
COLOR1.C	- Displays framed 2-color dither w/16 colors.
DC.C	- Sets foreground/background colors on display.
DRAW.C	- Draws colored dots on screen with mouse.
FRUIT.C	- Displays pre-stored IDA file on screen.
TEXT.C	- Displays text message on screen using font.

Appendix C

Example PASCAL Programs

These PASCAL programs are provided as examples for using the ISV Toolkit. The program listing is included, with comments. The compile, link and go batch file used is described in Section 4, Examples for PASCAL Library.

POINT.PAS - Draws a single point on screen

COLOR0.PAS - Displays framed 16 color palette on screen

COLOR1.PAS - Displays framed 2-color dither w/16 colors.

ADDENDUM TO

ISV TOOLKIT MANUAL

SECTION 5 (IDA)

IDA.EXE has been revised. The user should refer to IDA.DOC on the Utilities diskette for the most current information.