HEWLETT-PACKARD

Vectra System BIOS Technical Reference Manual

Notice

Personal Computer Group
974 East Arques Avenue
P.O. Box 486
Sunnyvale, CA 94086, U.S.A.

First Edition - Febuary 1988, Printed in Singapore
Part Number 45945-90012

Update Notice - March 1989

This package, which updates the original issue of the Vectra System BIOS Technical Reference Manual, provides BIOS update information for the Vectra QS/16, QS/20, RS/20C, and the RS/25C. With the replacement of the pages given, the Vectra System BIOS Technical Reference Manual is valid for the HP Vectra ES, QS, and RS series of personal computers. (Changes that have been made are explained.)

All references to the Vectra RS are also valid for the equivalent speed Vectra QS. The only difference between the BIOS of the Vectra QS and the BIOS of the Vectra RS is that the QS has a different PC ID (identification) flag. See Page B-19 for details.

Product Kit No. (manual and binder): 45945-60031
Manual Part No.: 45945-90012
First Update Part No. 5959-6796
Second Update Part No.: 5959-9816

Personal Computer Group
974 East Arques Avenue
P.O. Box 486
Sunnyvale, CA 94086, U.S.A.

Insert or replace the update pages in this package in the appropriate chapters of your Vectra System BIOS Technical Reference Manual.

CHANGE PAGES

Cover Page
Second line, under the bar
Replace: For the HP Vectra Series of Personal Computers
With: For the HP Vectra Series (ES, QS, RS) of Personal Computers

Page 1-1
Paragraph 1, second line
Replace: ES and RS series
With: ES, QS, and RS series

Page 1-1
Paragraph 2, third line
Replace: HP Vectra RS series
With: HP Vectra QS and RS series

Page 2-1
Paragraph 1, third line
Replace: as well as the Vectra ES and RS series discussed
With: as well as to the HP Vectra series of PCs discussed

Page 4-39
Paragraph 2, last line
REPLACE: all Vectra ES and RS series computers.)
or REPLACE: all HP Vectra series computers.)
With: all ES, QS, and RS Vectra series computers.)

Page 4-47
Move the last 3 row items in the table, starting with:
SF___GET___DEVTBL
SF___SET___DEVTBL
SF___DEF___DEVTBL
Between these row items in the table:
SF___CRV___REPORT___NAME and
F___PUT___BYTE

Page 4-57
Paragraph 1, second and third lines
Replace: For the Vectra ES and RS series computers
With: For the HP Vectra series of computers

Page 5-1
Paragraph 1, fourth and fifth lines (in the second bulleted item)
Replace: used with *both* the HP Vectra ES and RS series
With: used with the HP Vectra *series*

Page 5-13
Table 5-7, under <Pause>
Move: **Enhanced Keyboard only**
To: under "Action," instead of under "Key Combinations"

Page 5-13
Table 5-7, under <Ctrl>-<Alt>-<+>
Move: **Keyboard/DIN only**
To: under "Action", instead of under "Key Combinations"

Page 5-13
Table 5-7, within <Ctrl>-<Alt>-< > (across from "This key sequence toggles the computer speed.")
Add: a backslash (\ ) to the last part of this "Key Combination."

Page 5-13
Table 5-7, across from <Ctrl>-<Alt>-< >
Replace: On the Vectra RS this is handled by the system BIOS.
With: On the Vectra QS and RS this is handled by the system BIOS.

Page 5-13
Table 5-7, under <Shift>-<Print Screen>
Move: **Keyboard/DIN only**
Under: "Action," instead of under "Key Combinations"

Page 5-14
Table 5-8, across from F16___GET___EXT___KEY
Replace: buffer (Vectra ES and RS keycodes)
With: buffer (including new Vectra ES, QS, and RS keycodes)

Page 5-14
Table 5-8, across from F16___EXT___STATUS
Replace: Vectra ES and RS keycodes
With: Vectra ES, QS, and RS keycodes

Page 5-16
Paragraph 1, second and third lines
Replace: Vectra ES and RS series computers.
With: Vectra series of computers.

Page 5-16
Paragraph 2 [starting with F16___STATUS (AH=01H)], fifth line
Replace: HP VEctra ES and RS series
With: HP Vectra series

Page 5-19
Paragraph 1, second line
Replace: HP Vectra ES and RS series personal computers
With HP Vectra series of computers

Page 5-19
Paragraph that starts with AH
Change spelling of: Concatinated
To: Concatenated

Page 5-22
Within the CAUTION statement
Replace: "it should be aware that STD-BIOS"
With: "the programmer should be aware that the EX-BIOS"

Page 5-26
Paragraph 1, second line
Replace: ES and RS series computers
With: series of computers

Page 5-26
Paragraph 3 (starting with "On Exit")
After the line: BX = 0BH for low speed (see following table)
Add the line: 12H for medium speed (see following table)

Page 5-26
Table 5-12
Replace: table's title, "HP Vectra ES and RS Speeds"
With: new table title, "Speeds for HP Vectra Series of Computers"

Page 5-26
Table 5-12
Between: the "High" column heading and the "Low" column heading
Add: a new column heading, "Medium"

Page 5-26
Table 5-12
Under: the new column heading of "Medium"
Add, for the first four entries, dashes: " -- "

Page 5-26
Table 5-12, under the column for "Vectra"
Replace: RS/16
With: QS/16, RS/16

Page 5-26
Table 5-12, under the column for "Vectra"
Replace: RS/20
With: QS/20, RS/20

Page 5-26
Table 5-12
(1) At the end of the column for "Vectra," add a first new row for: RS/20C
(2) At the end of the column for "Vectra," add a second new row for: RS/25C

Page 5-26
Table 5-12
(1) Under the column for "High," and across from RS/20C, add: 20 MHz
(2) Under the column for "High," and across from RS/25C, add: 25 MHz

Page 5-26
Table 5-12
(1) Under the new column for "Medium," and across from RS/20C, add: 10 MHz
(2) Under the new column for "Medium," and across from RS/25C, add: 12.5 MHz

Page 5-26
Table 5-12
(1) Under the column for "Low," and across from RS/20C, add: 5 MHz
(2) Under the column for "Low," and across from RS/25C, add: 5 MHz

Summary of changes to Table 5-12 appear as follows:

Table 5-12. Speeds for HP Vectra Series of Computers

Vectra	High	Medium	Low
ES	8 MHz	-	8 MHz
ES/12	12 MHz	-	8 MHz
QS/16, RS/16	16 MHz	-	8 MHz
QS/20, RS/20	20 MHz	-	8 MHz
RS/20C	20 MHz	10 MHz	5 MHz
RS/25C	25 MHz	12.5 MHz	5 MHz

Page 5-27
Paragraph 1, first line
Replace: HP Vectra ES and RS series
With: HP Vectra series of computers,

Page 5-27
Paragraph 1, starting with F16___GET___INT___NUMBER (AX = 6F0DH)
After: the last line, "Registers Altered: AX"
Add the following paragraphs:

F16___SET___CACHE___ON (AX = 6F0FH) -- This subfunction enables memory caching.

```
On Entry:  AX = F16_SET_CACHE_ON (6F0FH)
On Exit:   AH = 00H (Successful)
                FEH (Cache subsystem is bad)
Registers Altered:  AX
```

F16___SET___CACHE___OFF (AX = 6F10H) -- This subfunction disables memory caching.

```
On Entry:  AX = F16__SET__CACHE__OFF (AX = 6F10H)
On Exit:   AH = 00H (Successful)
Registers Altered:  AX
```

F16___GET___CACHE___STATE (AX = 6F11H)
This subfunction returns the memory cache subsystem's state.

```
On Entry:  AX = F16__GET__CACHE__STATE (AX = 6F11H)
On Exit:   AH = 00H (Successful)
           AL bit 0 = 0 (Cache Disabled)
                    = 1 (Cache Enabled)
Registers Altered:  AX
```

F16___SET___MEDIUM___SPEED (AX = 6F12H)
This subfunction sets the computer's speed to medium.

```
On Entry:  AX = F16__SET__MEDIUM__SPEED (6F12H)
On Exit:   AH = 00H (Successful)
Registers Altered:  AX
```

Page 5-54
Table 5-19, across from 0D0H, under "Description," 3rd sentence,
Replace: "See Table 5-22 for bit definitions."
With: "See Table 5-21 for bit definitions."

Page 5-54
Table 5-19, across from 0D1H, under "Description," 3rd sentence,
Replace: "The bit definitions for this port are given in Table 5-22."
With: "The bit definitions for this port are given in Table 5-21."

Page 5-59
Paragraph 3, first and second lines
Replace: Refer to the *Vectra Hardware Technical Reference Manual* (for the ES or RS series)
With: Refer to Figures 5-2 and 5-3

Page 5-61
Paragraph 2, first and second lines
Replace: Refer to the *Vectra Hardware Technical Reference Manual* (for the ES or RS series)
With: Refer to Figures 5-2 and 5-3

Page 5-63
Paragraph 2, first and second lines
Replace: Refer to the *Vectra Hardware Technical Reference Manual* (for the ES or RS series)
With: Refer to Figures 5-2 and 5-3

Page 6-3
Paragraph 1, third line
Replace: (for either the HP Vectra ES, or RS personal computer)
With: (for the HP Vectra ES, QS, or RS personal computers).

Page 6-15
Paragraph 11, which starts with "Example," first line
Replace: (AH = 6F00H)
With (AX = 6F00H)

Page 7-2
Paragraph 5, (starting with "The flexible disc operation"), fourth lines and following
Replace:
For Vectra RS system, support for two additional flexible discs is achieved with a special Flexible Disc Expander card, if you have such a card installed the contents of the operation table are expanded, see Tables 7-1 and 7-1a.

With:
For the Vectra RS system only, support for two additional flexible discs is achieved with a special Flexible Disc Expander card. (If this card is installed, the contents of the operation table are expanded.) See Tables 7-1 and 7-1a.

Page 8-1
Paragraph 4, first line
Replace: (double word on **Vectra RS series**)
With: (double word on **Vectra QS and RS series)**

Page 8-8
Half-way through page
Replace: FOR HP VECTRA RS SERIES COMPUTERS
With: FOR HP VECTRA QS AND RS SERIES COMPUTERS

Page 8 8
Two-thirds through page
Replace: On Exit: EAX = Double word with all equipment information.
With: On Exit: EAX = Double word with all equipment information. (* Indicates for Vectra RS only.)

Page 8-8
In: the table under EAX = Double word with all equipment information
Replace throughout: Weitek 1167
With: Weitek 1167*

Page 8-18
3rd line from the bottom
Replace: 0FFH = Printer, timeout required.
With: 0FEH = Printer, timeout required.

Page 9-3
Second-to-last bulleted item
Replace: Test the coprocessor if present (80387 and Weitek coprocessor for Vectra RS series).
With: Test the coprocessor if present (80387 for Vectra QS series, and 80387 and Weitek coprocessor for Vectra RS series).

Page 9-3
After: the sixth bulleted item, "Test the first 64 KB of system RAM."
Include: a new bulleted item, "Test memory cache subsystem (Vectra RS/20C and RS/25C.)"

Page 9-3
Before: the last bulleted item, "Test serial and parallel port (parallel port not tested in Vectra RS series)."
Include: a new bulleted item, "Test the CPU clock speed."

Page 9-3
Change: the last bulleted item
From: "Test serial and parallel port (parallel port not tested in Vectra RS series)."
To: "Test serial port."

Reorder the bulleted items so they have the following order:

6. Initialize the video display for diagnostic messages.
1. Test the operation of the CPU.
2. Test the system ROM.
3. Test and initialize 8254 timer/counter and start the refresh counter.
7. Test and initialize DMA controllers and DMA page registers.
4. Test the first 64 KB of system RAM.
5. Test memory cache subsystem (Vectra RS/20C and RS/25C only.)
8. Test and initialize the 8259A interrupt controllers.
9. Test the 8042 controller and Scandoor.
10. Test the HP-HIL controller.
11. Test CMOS RAM for integrity.
12. Determine if manufacturing electronic tool is present. If so, run manufacturing test.
13. Test the remaining base system RAM (RAM above the first 64 KB).
14. Test the extended RAM above memory address 100000H (protected mode RAM.)
15. Test the real-time clock portion of the RTC/CMOS chip.
16. Test the keyboard interface and the keyboard itself.
17. Test the flexible disc controller subsystem.
18. Test the coprocessor if present (80287 for Vectra ES series, 80387 for Vectra QS series, and 80387 and Weitek coprocessor for Vectra RS series). 19. Test the CPU clock speed.
20. Test serial port.

Page 9-3
Summary -- the bulleted items will now appear as follows:

- Test the operation of the CPU.
- Test the system ROM.
- Test and initialize 8254 timer/counter and start the refresh counter.
- Test the first 64 KB of system RAM.
- Test memory cache subsystem (Vectra RS/20C and RS/25C only.)
- Initialize the video display for diagnostic messages.
- Test and initialize DMA controllers and DMA page registers.
- Test and initialize the 8259A interrupt controllers.
- Test the 8042 controller and Scandoor.
- Test the HP-HIL controller.
- Test CMOS RAM for integrity.
- Determine if manufacturing electronic tool is present. If so, run manufacturing test.
- Test the remaining base system RAM (RAM above the first 64 KB).
- Test the extended RAM above memory address 100000H (protected mode RAM.)
- Test the real-time clock portion of the RTC/CMOS chip.
- Test the keyboard interface and the keyboard itself.
- Test the flexible disc controller subsystem.
- Test the coprocessor if present (80287 for Vectra ES series, 80387 for Vectra QS series, and 80387 and Weitek coprocessor for Vectra RS series).
- Test the CPU clock speed.
- Test serial port.

Page 9-3
Last line of page
Replace: (for **Vectra RS series**).
With: (for **Vectra QS and RS series**).

Pages 9-4 to 9-9
Title of Table 9-2a
Replace: Vectra ES Series POST
With: Vectra ES POST

Pages 9-10 to 9-15
Title of Table 9-2b
Replace: Vectra RS Series POST
With: Vectra QS and RS POST

Pages 9-10 to 9-15
Delete: entire column entitled "Chip" and all entries underneath

Page 9-11
Table 9-2b.
After row for: 0709
Include this row information:
Code: 070B
Test: 82C301
Description: CPU clock too slow at MEDIUM speed.

Page 9-11
Table 9-2b.
After new row for: 070B
Include this row information:
Code: 070C
Test: 82C301
Description: CPU clock too fast at MEDIUM speed.

Page 9-15
Table 9-2b, thoughout, under the "Description" column
Replace: Weitek
With: Weitek *

Page 9-15
Across from: "AF00" and "Weitek"
Under the column: Description
Change: Weitek coprocessor (COP) Test failed to enter Protected Mode.
To: Weitek* coprocessor (COP) Test failed to enter Protected Mode. (* indicates for Vectra RS only.)

Page 9-15
Table 9-2b
After: row for AF0C
Add: the following row information for B300 through BFFF:

Code: B300
Test: 8042 **
Description: Failed to switch to protected mode. (** **indicates errors detected by the Memory Cache Test.)**

Code: B301-B307
Test: 82385
Description: General cache subsystem failure.

Code: B400-B7FF
Test: Main Memory **
Description: Read/write test of DRAM locations 60000h-6FFFFh failed.
Decode bits in error code to isolate failing memory module:
BXYZ where
X = 01aa => aa specifies which byte is bad (0-3)
YZ = bbbb bbbb => b=1 specifies bad bit
e.g.: 0100 0010 => bits 6 and 1 bad

Code: B800-BBFF
Test: Static RAM
Description: Read/write test of SRAM failed.
Decode bits in error code to isolate failing chips:
BXYZ where
X = 10aa => aa specifies which byte is bad (0 - 3)
YZ = bbbb bbbb => b=1 specifies bad bit
e.g.: 0100 0010 => bits 6 and 1 bad

Code: BC00-BFFF
Test: Static RAM
Description: Marching ones test of SRAM failed.
Decode bits in error code to isolate failing chips:
BXYZ where
X = 11aa => aa specifies which byte is bad (0 - 3)
YZ = bbbb bbbb => b=1 specifies bad bit
e.g.: 0100 0010 => bits 6 and 1 bad

Page 9-15/9-16
First sentence after Table 9-2b, starting with "If the POST process is initiated.. "
Replace: "If the POST process is initiated by a soft reset, the RAM tests are not executed."
With: "If the POST process is initiated by a soft reset, the RAM tests and the cache memory test are not executed."

Page 9-19
Paragraph 4, starting with "Shadow RAM." Title, and first and fourth lines
Replace: HP Vectra RS
With: HP Vectra QS and RS

Page A-6
Table A-2
Across from: INT Hex code 16H
Add: the following row information after the row for Function Value 02H --

Function Value: 03H
Function Equate: F16___SET___TYPE___RATE
Definition: Set typematic rates.

Function Value: 05H
Function Equate: F16___PUT___KEY
Definition: Put data into keyboard buffer.

Function Value: 10H
Function Equate: F16___GET___EXT
Definition: Read keycode from buffer (including extended keycodes).

Function Value: 11H
Function Equate: F16___EXT___STATUS
Definition: Report extended keyboard status

Function Value: 12H
Function Equate: F16___EXT___KEY___STATE
Definition: Get Extended Key Modifier status.

Page A-6
Table A-2.
Across from: INT Hex code 17H
For: the Function Value 6F01H
Change: the Function Equate, "F17___READ___STATUS"
To: a blank line

Page A-6
Table A-2.
Across from: INT Hex code 17H
For: the Function Value 6F03H
Change: the Function Equate, "F17___GET___BUFFER"
To: a blank line

Page A-6
Table A-2.
Across from: INT Hex code 17H
After: the Function Value 6F04H
Add: the following row information --

Function Value: 6F0FH
Function Equate: F16___SET___CACHE___ON
Definition: Turn cache on.

Function Value: 6F10H
Function Equate: F16___SET___CACHE___OFF
Definition: Turn cache off.

Function Value: 6F11H
Function Equate: F16___GET___CACHE___STATE
Definition: Get current cache state.

Function Value: 6F12H
Function Equate: F16___SET___MEDIUM___SPEED
Definition: Sets medium speed for cache machines.

Page A-15
Table A-3
Delete the following rows of information:

0114H	04/16	SF_KEYBOARD_REPEAT	Set typematic values
0114H	04/18	SF_KEYBOARD_LED	Set keyboard LED states
0114H	06	F_PUT_BYTE	Write one byte to specified HP-HIL device

Page A-15
Table A-3
In the place of the deleted rows above, include the following rows of information:

Vector Address	Func. Value	Function Equate	Definition
0114H	04/20	SF_GET_DEVTBL	Gets physical device table address
0114H	04/22	SF_SET_DEVTBL	Sets physical device table address
0114H	04/24	SF_DEF_DEVTBL	Sets default physical device table

Page B-13
Under: Flexible Disc Expander Adapter Data Area

Replace:
This is only applicable in **Vectra RS** systems with the Flexible Disc Expander adapter card installed.

With:
This applies solely to the **Vectra RS** systems, and only when the Flexible Disc Expander adapter card is installed.

Page B-15
Item f, first line
Replace: HP Vectra ES and RS series computers
With: HP Vectra series of computers

Page B-15
Item f, fourth and fifth lines
Replace: RAM in the Vectra ES and RS computers
With: RAM in the Vectra series of computers

Page B-15
Item f, last line
Replace: of the Vectra RS
With: of the Vectra QS and RS

Page B-19
Top of page
Replace: table title, "Vectra ES and RS Series Processor Clock Rates"
With: new title, "Processor Clock Rates for HP Vectra Series of Computers"

Page B-19
Table at top of page, under the column for "Computer"
Replace: Vectra RS/16
With: Vectra QS/16, RS/16

Page B-19
Table at top of page, under the column for "Computer"
Replace: Vectra RS/20
With: Vectra QS/20, RS/20

Page B-19
Table at top of page,
Add: two new rows --

Vectra RS/20C	14H (20 MHz)	05H (5 MHz)
Vectra RS/25C	19H (25 MHz)	05H (5 MHz)

Page B-19
Table at top of page, under the column for "Clock Rate (High)"
Replace: 0CH (8 MHz)
With: 0CH (12 MHz)

Summary: New table on page B-19 appears as follows --

Processor Clock Rates for HP Vectra Series of Computers

Computer	Clock Rate (High)	Clock Rate (Low)
Vectra ES	08H (8 MHz)	08H (8 MHz)
Vectra ES/12	0CH (12 MHz)	08H (8 MHz)
Vectra QS/16, RS/16	10H (16 MHz)	08H (8 MHz)
Vectra QS/20, RS/20	14H (20 MHz)	08H (8 MHz)
Vectra RS/20C	14H (20 MHz)	05H (5 MHz)
Vectra RS/25C	19H (25 MHz)	05H (5 MHz)

Page B-19

```
As shown in the update below,
after the PC ID for the Vectra RS/16  00110

add the PC ID for the Vectra QS/16    00111
add the PC ID for the Vectra QS/20    01000
add the PC ID for the Vectra RS/20C   01001
add the PC ID for the Vectra RS/25C   01010
add the PC ID for the Vectra LS/12    01011
```

Also, on the next line,
Change: 00111 through 11111 - Reserved
or change: 01001 through 11111 - Reserved
To: 01100 through 11111 - Reserved

```
  Bits:

  7  6  5  4  3  2  1  0
  -------  -------------
     |           |------>  00000 - Original Vectra PC
     |                     00001 - Vectra ES/12
     |                     00010 - Vectra RS/20
     |                     00011 - Portable Vectra CS
     |                     00100 - Vectra ES
     |                     00101 - Vectra CS
     |                     00110 - Vectra RS/16
     |                     00111 - Vectra QS/16
     |                     01000 - Vectra QS/20
     |                     01001 - Vectra RS/20C
     |                     01010 - Vectra RS/25C
     |                     01011 - Vectra LS/12
     |       01100 through 11111 - Reserved
     |
     |------------------->   000 - 80286
                             001 - 8088
                             010 - 8086
                             011 - 80386
               100 through   111 - Reserved
```

Page B-19
Move: Paragraph entitled "Machine Capability Marker"
To: Page B-20

Page B-20
Move: Paragraph entitled "Year of the ROM BIOS Release (in BCD)"
and: Paragraph entitled "Week of the ROM BIOS Release (in BCD)"
To: Page B-21

(ADD: NEW page B-21)

Page C-6
2nd to last line
Replace: Note that Vectra ES and RS series computer
With: Note that Vectra series of personal computers

Page D-1
Paragraph 1, last two lines
Replace: *Vectra Accessories Technical Reference Manual* (for either the Vectra ES or RS series).
With: *Vectra Accessories Technical Reference Manual*

Page F-2
Last line
Replace: HP Vectra ES and RS
With: HP Vectra series of personal computers

Page F-3
Paragraph 3 (starting with "This code"), first line
Replace: HP Vectra ES and RS series
With: HP Vectra series of personal computers

Page F-3
Paragraph 3, second and third lines
Replace: unique features of the HP Vectra ES and RS series. This method
With: unique features of the HP Vectra series of personal computers. (However, this method

Page F-20
Paragraph 3 (starting with "The driver is"), fourth line
Replace: all Vectra ES and RS series
With: all Vectra series

Page Glossary 2
Coprocessor, second line
Replace: The 80287 (Vectra ES), 80387 and Weitek coprocessor (Vectra RS only) are
With: The 80287 (Vectra ES), the 80387 (Vectra QS and RS), and the Weitek coprocessor (Vectra RS only) are

Page Glossary 5
Original Vectra PC, first line
Replace: Vectra ES and RS series
With: Vectra ES, QS, and RS series

Index

Insert: Cache 5-27, 9-3, 9-15, 9-16, A-6

Delete: "Disk (see Disc)"

Replace: Machine Capability Marker B-19
With: Machine Capability Marker B-20

Replace: Week of BIOS Release B-20
With: Week of BIOS Release B-21

Replace: Year of BIOS Release B-20
With: Year of BIOS Release B-21

HP Vectra System BIOS Technical Reference Manual

For the HP Vectra Series (ES, QS, RS) of Personal Computers

Contents

Vectra System BIOS Technical Reference Manual

Chapter 1
Introduction

Chapter 2
ROM BIOS Overview

Chapter 3
Video

Chapter 4
Input System and HP-HIL

Chapter 5
Keyboard

Chapter 6
Serial and Parallel I/O

Chapter 7
Disc

Chapter 8
System Drivers

Chapter 9
System Processes

Appendix A
BIOS Interrupts

Appendix B
Memory Map

Glossary

References

1 Introduction

This manual contains a detailed description of the ROM Basic Input/Output System (BIOS) of the HP Vectra ES, QS, and RS series of personal computers. Entry points, including the industry standard ROM BIOS entry points and function calls, are documented in this manual.

This manual deals extensively with programming and programming concepts. It presumes that the reader is familiar with the Microsoft Macro Assembler (MASM), and the Intel iAPX 80286 (HP Vectra ES series) and iAPX 80386 (HP Vectra QS and RS series) processor architecture.

Terms Used In This Manual

In this manual, the term CPU (Central Processing Unit) will be used to refer to both the 80286 and 80386 processors when a function or operation described is exactly the same for both. Other abbreviations, acronyms, and terms used throughout this volume are listed in a glossary at the back of this volumn. Related documents which may be of interest to programmers and advanced users are also listed at the end of this volume in the "References" section.

System Software

Software operating on the system may be viewed as a three-level hierarchy: application programs, operating system, and ROM BIOS. These three levels are defined as follows:

Application Programs

An application program is the top level of software. It performs application-specific functions (i.e., spreadsheet or word processing functions). Application programs rely on either DOS or the ROM BIOS for system functions such as character or disc I/O.

Operating System

The operating system provides the control and support functions necessary for an application program to be executed. The operating system provides file-oriented functions, as well as providing basic support for character I/O.

ROM BIOS

The ROM BIOS provides the interface between operating system software and the hardware. The ROM BIOS provides a dual function; it constitutes the low level interface between the hardware and operating system, as well as providing extended functions to application programs.

The higher the software level, the more powerful the functions provided by the software. However, along with this power often comes additional overhead which reduces performance and flexibility. A system programmer should choose the level of software interface required by the individual set of design constraints. It is good programming practice to use the highest level of system software that gets the job done. Some system functions can be performed only on the highest level, since only system software supports the function. However, other system functions may be performed at more than one level. Using a lower level such as the ROM BIOS provides improved speed of execution and additional flexibility. Using ROM BIOS routines may affect program portability to future HP products, and to other industry-standard PCs.

The ROM BIOS provides a powerful set of system functions, allowing application programs full access to the capabilities of the system while maintaining a hardware-independent interface. The ROM BIOS also allows the programmer or system designer to tailor the system to a specific set of design constraints. Some of the tailoring methods provided to the programmer are:

- The number of interrupts can logically expand to fit requirements.
- Adapter cards can obtain a limited amount of RAM from the system BIOS without installing device drivers.
- Applications can expand the features of the keyboard without replacing the industry standard driver (INT 16H).

These methods maintain application compatibility with minimal effect on system performance.

2

ROM BIOS Overview

The ROM BIOS is divided into two components, the Standard BIOS (STD-BIOS) and the Extended BIOS (EX-BIOS). The STD-BIOS supports the industry standard set of BIOS functions The EX-BIOS is unique to the original HP Vectra PC as well as to the HP Vectra series of PCs discussed in this manual. It provides a wide range of system functions and support for HP peripherals. The STD-BIOS and the EX-BIOS are contained in the system ROM which resides at the top of system memory.

NOTE

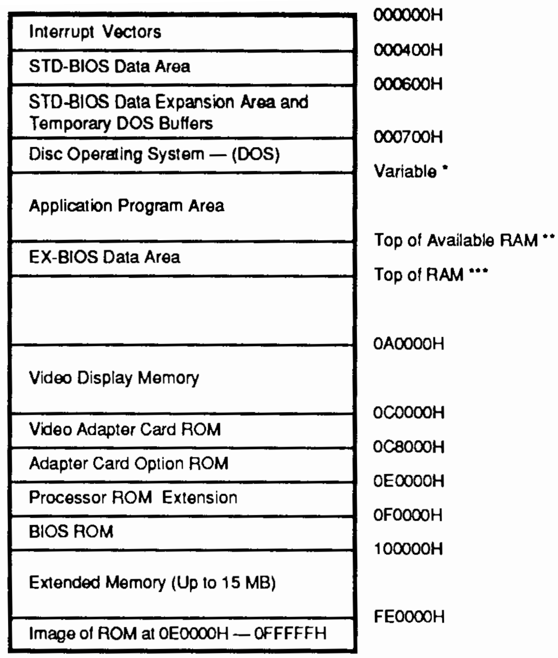
Throughout the remainder of this manual the terms ROM BIOS, STD-BIOS, and EX-BIOS will be used. STD-BIOS and EX-BIOS are defined above. The term ROM BIOS will be used to indicate the union of STD-BIOS and EX-BIOS. As mentioned before, the term CPU (Central Processing Unit) will refer to both the 80286 and 80386 series of processors.

This chapter contains an overview of the components of the ROM BIOS. These components are the interrupt (also called "INT") vectors, code modules, and data structures. Interrupt vectors form the link between the operating system, applications, and the ROM BIOS. The code modules perform the ROM BIOS functions. Data structures provide the means for the ROM BIOS (and to some extent the applications) to maintain driver variables, data buffers, etc.

Memory Locations

Code modules are accessed through interrupt vectors. The interrupt vectors reside in the first 1KB of system RAM. Usually a code module has an associated data structure. The data structures for the STD-BIOS code modules reside in system RAM in absolute memory locations 00400H through 005FFH. The data structures for the EX-BIOS code module reside at the top of system RAM. The address of the EX-BIOS data area will vary depending on the particular configuration of the system.

Figure 2-1 shows the components of the ROM BIOS and their location within the system memory. Each of the ROM BIOS components is discussed in detail in the remainder of this chapter.



* The length of the operating system is revision dependent.

** The Top of Available RAM is dependent on system configuration; in a 640 KB system it is usually 09F000H. Refer to the corresponding hardware TRM for more information.

*** The Top of RAM is dependent on system configuration; in a 640 KB system it is 09FFFFH. Refer to the corresponding hardware TRM for more information.

Figure 2-1. Memory Map Block Diagram

Interrupts

The interface to the ROM BIOS is through the interrupt structure of the CPU. The system allows for three types of interrupts.

- Processor Interrupts--These interrupts allow system software to recover from error conditions and other hardware exceptions.
- Hardware Interrupts--These interrupts are generated by two compatible (8259A) interrupt controllers integrated into a VLSI chip (P/N 82C206) located on the Processor PCA. Hardware interrupts indicate that a system hardware component or peripheral requires service.
- Software Interrupts--These interrupts are generated through the software "INT n" instruction. Software interrupts allow system functions to be quickly and easily called by any program.

Interrupt vectors for the processor interrupts are defined by the CPU. Interrupt vectors for the hardware interrupts are mapped by the values programmed into the 8259A interrupt controllers which are initialized by the ROM BIOS. Processor and/or hardware interrupts may be simulated by a software interrupt mapped to the same interrupt vector. For example, Interrupt 0 is mapped by the CPU for Divide-by-0 error. The service routine for this error condition may be executed by an INT 0 instruction.

Each interrupt has an interrupt vector associated with it. The interrupt vector contains the Code Segment and Instruction Pointer of the service routine for that interrupt. Each of these vectors consists of two words (four bytes). The CPU architecture supports 256 interrupt vectors which occupy the first 1024 bytes (00000H-003FFH) of system memory.

The interrupt vectors maintain industry standard compatibility while offering the expanded capabilities of the HP EX-BIOS functions. Table 2-1 lists the interrupt vector assignments.

In order for the system to function properly, processor and hardware interrupt vectors are initialized to valid service routines. Most unused vectors point to a null routine in the BIOS, which issues an End-of-Interrupt (EOI) signal to the 8259A interrupt controllers (when required) and returns. The Keyboard Break and Timer Tick software interrupt vectors point to an interrupt return (IRET) instruction in the BIOS. These vectors are indicated by an IRET in Table 2-1. Several software vectors are used as pointers to data blocks instead of interrupt service routines. These vectors are indicated by an interrupt vector used as a pointer to data (PT) in Table 2-1.

Table 2-1. Interrupt Vector Assignments

INT	Address	Function	Type/ Routine*	Service
0	000-003H	Divide by Zero	PI (1)	STD-BIOS
1	004-007H	Single Step	PI (1)	STD-BIOS
2	008-00BH	Nonmaskable Interrupt	PI	STD-BIOS
3	00C-00FH	Breakpoint	PI (1)	STD-BIOS
4	010-013H	Arithmetic Overflow	PI (1)	STD-BIOS
5	014-017H	Print Screen	SW (2)	STD-BIOS
6	018-01BH	Invalid Opcode	PI (1)	STD-BIOS
7	01C-01FH	Reserved	PI (1)	STD-BIOS
8	020-023H	Timer Interrupt	HW	
9	024-027H	Keyboard ISR (IRQ 1)	HW	STD-BIOS
A	028-02BH	Reserved (IRQ 2)	HW	STD-BIOS
B	02C-02FH	Serial Port 1 ISR (IRQ 3)	HW (1)	STD-BIOS
C	030-033H	Serial Port 0 ISR (IRQ 4)	HW (1)	STD-BIOS
D	034-037H	Printer Port 2 ISR (IRQ 5)	HW (1)	STD-BIOS
E	038-03BH	Flexible Disc ISR (IRQ 6)	HW	STD-BIOS
F	03C-03FH	Printer Port 1 ISR (IRQ 7)	HW (1)	STD-BIOS
10	040-043H	Video	SW (2)	STD-BIOS
11	044-047H	Equipment Check	SW (2)	STD-BIOS
12	048-04BH	Memory Size	SW (2)	STD-BIOS
13	04C-04FH	Flexible Disc/ Hard Disc	SW (2)	STD-BIOS
14	050-053H	Serial	SW (2)	STD-BIOS
15	054-057H	System Functions	SW (2)	STD-BIOS
16	058-05BH	Keyboard	SW (2)	STD-BIOS
17	05C-05FH	Printer	SW (2)	STD-BIOS
18	060-063H	Reserved	SW (3)	STD-BIOS
19	064-067H	Boot	SW (2)	STD-BIOS

Table 2-1. Interrupt Vector Assignments (Cont.)

INT	Address	Function	Type/ Routine*	Service
1A	068-06BH	Time and Date	SW (2)	STD-BIOS
1B	06C-06FH	Keyboard Break	SW (3)	STD-BIOS
1C	070-073H	Timer Tick	SW (3)	STD-BIOS
1D	074-077H	Video Parameter Table	PT	STD-BIOS
1E	078-07BH	Flexible Disc Parameter Table	PT	STD-BIOS
1F	07C-07FH	Graphics Character Table	PT	STD-BIOS
20	080-083H	Program Terminate	SW	DOS
21	084-087H	DOS Function Calls	SW	DOS
22	088-08BH	DOS Terminate Address	PT	DOS
23	08C-08FH	DOS <CTRL>-<Break> Address	SW	DOS
24	090-093H	DOS Critical Error	SW	DOS
25	094-097H	DOS Absolute Disc Read	SW	DOS
26	098-09BH	DOS Absolute Disc Write	SW	DOS
27	09C-09FH	DOS Terminate Stay Resident	SW	DOS
28-32	0A0-0CBH	Reserved for DOS	SW	DOS
33	0CC-0CFH	Mouse (RAM driver)	SW (2)	N/A
34-3F	0D0-0FFH	Reserved for DOS	SW	DOS
40	100-103H	Alternate Flexible Disc	SW	STD-BIOS
41	104-107H	Hard Disc Parameter Table (0)	PT	STD-BIOS
42-45	108-117H	Reserved	SW	STD-BIOS
46	118-11BH	Hard Disc Parameter Table (1)	PT	STD-BIOS
47-5F	11C-17FH	Reserved	SW	STD-BIOS
60-67	180-19FH	Reserved for User Programs Programs	SW	N/A
68-6E	1A0-1BBH	Unused	SW	N/A
6F	1BC-1BFH	Default EX-BIOS Entry Point	SW (2)	EX-BIOS
70	1C0-1C3H	Real-time Clock ISR (IRQ 8)	HW	STD-BIOS
71	1C4-1C7H	SW Redirected (IRQ 9)	HW	STD-BIOS
72	1C8-1CBH	Reserved (IRQ 10)	HW (1)	STD-BIOS

Table 2-1. Interrupt Vector Assignments (Cont.)

INT	Address	Function	Type/ Routine*	Service
73	1CC-1CFH	Reserved (IRQ 11)	HW (1)	STD-BIOS
74	1D0-1D3H	HP-HIL (default IRQ 12)	HW	EX-BIOS
75	1D4-1D7H	Coprocessor (IRQ 13)	HW	STD-BIOS
76	1D8-1DBH	Hard Disc ISR (IRQ 14)	HW	STD-BIOS
77	1DC-1DFH	Reserved (IRQ 15)	HW (1)	STD-BIOS
78-7F	1E0-1FFH	Not Used	SW	N/A
80-F0	200-3C3H	Reserved	SW	N/A
F1-FF	3C4-3FFH	Not Used	SW	N/A

* PI--Processor interrupt
HW--Hardware interrupt
SW--Software interrupt
PT--Interrupt vector used as pointer to data
N/A--Not applicable

(1) UI--Unused interrupt ISR
(2) DRVR--Application callable entry point
(3) IRET--Interrupt return

ROM BIOS Drivers and Functions

The ROM BIOS is comprised of many drivers. For example, there is a driver to perform video functions, one to perform disc functions, etc. The ROM BIOS drivers are organized into two components. One component contains the STD-BIOS drivers that support the STD-BIOS functions. The second component contains EX-BIOS drivers that support unique HP features.

Each driver supports one or more functions. A function can be viewed as a specific task. For example, the Video Driver supports 22 separate functions that perform tasks such as setting the display mode, moving the cursor, and displaying characters.

STD-BIOS Drivers

Drivers in the STD-BIOS are accessed through an interrupt. STD-BIOS drivers are accessed through interrupts 05H and 10H through 1CH. Drivers are accessed by performing a software INT n instruction, where n is the interrupt number assigned to the driver (refer to Table 2-1.)

The function code and any required data are passed in the CPU registers. Data passing conventions for STD-BIOS drivers vary; however, there are aspects which are common.

- Most of the STD-BIOS drivers support more than one function. Therefore, multi-function drivers must have the desired function code passed as part of the data. The AH register is used on all multi-function drivers to pass the function code.

- Byte and word data are passed in the internal registers of the CPU. Registers AL, BX, CX, and DX are usually used for this purpose. The register assignments and number of registers used depend on the driver and driver function.

- If the amount of data cannot fit in the internal registers of the CPU, a data buffer in system memory is used. This buffer is usually pointed to by ES:BX, ES:BP, or ES:SI.

- Drivers may modify one or more registers. The registers which are maintained and the registers which are modified vary from driver to driver. The registers which are modified are listed in each function description.

Calling STD-BIOS Drivers

The following program example demonstrates how a typical STD-BIOS driver is accessed. The function sets the position of the cursor on display page 0 to row 20, column 10. The function code (02H) is passed in register AH. The row position, the column position, and the page number are passed respectively in DH, DL, and BH.

```
MOV  AH,02H  ;Function number
MOV  DH,14H  ;Row number (Row 20)
MOV  DL,0AH  ;Column number (Column 10)
MOV  BH,0H   ;Page number
INT  10H     ;Call Video driver
```

The STD-BIOS drivers support all industry standard BIOS functions. In addition, many of the drivers have functions that support enhanced features. These functions are referred to as "HP extensions" throughout the remainder of this manual. These enhancements are accessed through function code (default 06FH) of their respective driver. Most of these extended functions are further divided into subfunctions. For example, the HP extended function for the Video driver has six subfunctions which allow access to the enhanced features of the Multimode Video Display Adapter. The function code (06FH) is placed in the AH register and the subfunction code is placed in the AL register for all HP extensions.

The following program example uses HP extensions to turn off the HP cursor control keypad on the Vectra Keyboard/DIN **(this keyboard is available for Vectra ES series computers only)**.

```
MOV  AH,6FH  ; HP Function
MOV  AL,07H  ; Switch Keyboard
MOV  BL,02H  ; Disable CCP: Turn Cursor Control Pad Off
INT  16H     ; Call Keyboard Driver
```

We suggest you verify that HP extensions to each STD-BIOS driver are available prior to actually calling them. This is accomplished through subfunction 0 on each driver. An example of this can be found in Chapter 3 of this manual under the F10_INQUIRE (AX = 6F00H) function description.

EX-BIOS Drivers

The EX-BIOS drivers provide a wide range of functions not found in the STD-BIOS drivers. The EX-BIOS drivers are accessed through a a software interrupt vector called the "HP_ENTRY" interrupt (default 06FH). **Since this interrupt number can change from its default, a routine called "CALL SYSCALL" should be used in its place. This routine finds and calls the correct HP interrupt number.**

Due to the large number of EX-BIOS drivers, it would be impossible to give each driver its own interrupt vector and still maintain industry standard compatibility. Therefore, each driver is assigned its own number, which is placed in the BP register.

The CALL SYSCALL Routine

The following shows how the CALL SYSCALL routine works:

```
;------ SYSCALL
;
;       Issue an HP system call. This routine assumes that the EX-BIOS
;       is enabled.
;
;       When first called, this routine will patch the first instruction
;       "JMP SHORT PATCH" to become "INT XXH" where XXH is the current HP
;       interrupt number.
;

SYSCALL PROC    NEAR
        JMP     SHORT PATCH             ; Patch the jump if first time.
        RET

PATCH:
        PUSH    AX
        MOV     AX,6F0DH                ; Get current interrupt.
        INT     16H                     ; Extended INT 16H call.
        CMP     AH,2                    ; Is it unsupported.
        JNE     PATCH2                  ; No, AH is the interrupt number.
        MOV     AH,6FH                  ; Assume default 6FH.

PATCH2:
        MOV     AL,0CDH                 ; INT instruction opcode.
        MOV     WORD PTR [SYSCALL],AX   ; Patch JMP   SHORT PATCH above.
        POP     AX                      ; Recover used register.
        JMP     SYSCALL                 ; Perform the call.

SYSCALL ENDP
```

Calling EX-BIOS Drivers

As with the STD-BIOS drivers, each EX-BIOS driver may support one or more functions. A function code placed in the AH register selects the desired function within the driver. In addition, a subfunction code passed in the AL register is required by many EX-BIOS functions.

The following program example demonstrates access to a typical EX-BIOS driver. The function executes a "beep" on the speaker.

```
MOV   AH,3AH     ; Function: F_SND_BEEP
MOV   BP,12H     ; Driver Name: V_SYSTEM
PUSH  DS         ;
CALL  SYSCALL    ; Call EX-BIOS driver
POP   DS         ;
```

On leaving the EX-BIOS driver the BP and DS registers will be modified while the AH register usually contains the return status of the driver call.

It is good programming practice to verify that the EX-BIOS is accessible, and to identify the HP interrupt number (once) prior to actually calling it by using the "CALL SYSCALL" routine.

EX-BIOS Standard Functions

Many EX-BIOS drivers support a standard set of functions and subfunctions as listed in Table 2-2. While these functions and subfunctions are defined, it is not required that they all be implemented by every driver. In addition, EX-BIOS drivers may implement functions other than those listed. Most EX-BIOS drivers use a standard set of return status codes reported in the AH register at the completion of a driver's function call. Some of these return status codes and their definitions are listed in Table 2-3. A driver may return status code of RS_UNSUPPORTED (02H) for a given function.

Function codes and return statuses are described in detail in Appendix G.

Table 2-2. EX-BIOS Defined Functions

Function Subfunction	Definition	Register AH AL
F_ISR	Responds to a logical Interrupt Service Request (ISR).	00
F_SYSTEM	Executes one of several standard subfunctions.	
SF_INIT	Starts the initialization of a driver.	02 00
SF_START	Completes the initialization process of the driver.	02 02
SF_REPORT_STATE	Reports the state of the driver.	02 04
SF_VERSION_DESC	Reports the revision number and date code of the driver.	02 06
SF_DEF_ATTR	Reports the default configuration of the driver.	02 08
SF_GET_ATTR	Reports the current configuration of the driver.	02 0A
SF_SET_ATTR	Overrides the current configuration of the driver.	02 0C

Table 2-2. **EX-BIOS Defined Functions** (Cont.)

Function Subfunction	Definition	Register AH AL
SF_OPEN	Reserves the driver for exclusive access. Requests any resources required by the driver.	02 0E
SF_CLOSE	Releases the driver from exclusive access.	02 10
SF_TIMEOUT	Reports to the driver that a requested timeout has occurred.	02 12
SF_INTERVAL	Reports to the driver that a requested 60 Hz interval has expired.	02 14
SF_TEST	Performs a hardware test.	02 16
F_IO_CONTROL	Executes the following subfunctions and any driver-dependent subfunctions.	
SF_LOCK	Reserves the sub-address device specified for exclusive access.	04 00
SF_UNLOCK	Releases the sub-address specified from the exclusive access.	04 02
F_PUT_BYTE	Writes a byte of data.	06
F_GET_BYTE	Reads a byte of data.	08
F_PUT_BUFFER	Writes a variable-length buffer of data (supported by character devices).	0A
F_PUT_BLOCK	Writes a fixed-length buffer of data (supported by block devices).	0B
F_GET_BUFFER	Reads a variable-length buffer of data (supported by character devices).	0C
F_GET_BLOCK	Reads a fixed-length block of data (supported by block devices).	0C
F_PUT_WORD	Writes a word of data.	0E
F_GET_WORD	Reads a word of data.	10

EX-BIOS Parameter Passing Conventions

When calling EX-BIOS drivers, the function code is placed in the AH register, and the subfunction code (if any) is placed in the AL register. Note that the function and subfunction codes are multiples of two in order to facilitate decoding by the drivers.

The general parameter passing conventions used by the EX-BIOS drivers are also defined. These register conventions are as follows:

```
On Entry: BP = V_DRIVER_NAME
          AH = F_FUNC_CODE
          AL = SF_FUNC_CODE (if required by driver)
          CX = On write: byte count (if required by driver)
               On read: maximum permissible byte count
               (if required by driver)
       ES:DI = Buffer pointer or context area (if required by driver)

On Exit:  AH = Return status
          CX = On read: byte count (if required by driver)
               On write: number of bytes written (if required by driver)
       ES:DI = Buffer pointer or context area (if required by driver)
       DS,BP   Always modified (unless otherwise indicated)
```

EX-BIOS Return Status Codes

EX-BIOS drivers are expected to report a Return Status Code upon completion. This code is returned in the AH register. Several return status codes have been defined in Table 2-3.

Table 2-3. EX-BIOS Return Status Codes

Return Status Variable	Return Status Code	Indication
RS_SUCCESSFUL	000H	The requested function executed correctly.
RS_UNSUPPORTED	002H	The requested function or subfunction is not implemented or is unsupported.
RS_FAIL	0FEH (-02H)	The driver failed the operation in an error state.
RS_BAD _PARAMETER	0FAH (-06H)	The driver received a bad parameter.
RS_BUSY	0F8H (-08H)	The requested driver is busy.
RS_NO_VECTOR	0F6H (-OAH)	EX-BIOS Vector table is out of RAM or room for more drivers.
RS_OFFLINE	0F4H (-OCH)	Device is offline.
RS_OUT_OF_PAPER	0F2H (-OEH)	Device is out of paper.

If additional drivers are installed in the system, they should conform to the defined statuses wherever possible. However, to maintain coding efficiency and/or functional accuracy, a driver may create a return status other than those listed in Table 2-3.

NOTE

Return status conditions are always multiples of two. Negative return status codes indicate error conditions, while positive status codes indicate exceptional conditions to the caller. For example, the status code RS_UNSUPPORTED indicates the driver does not support a function which may or may not be an error, while RS_OUT_PAPER requires some kind of response by the caller.

Data Structures

BIOS drivers require RAM data area to perform their functions. The layout and placement of the data areas for the STD-BIOS and EX-BIOS drivers differ. This is discussed in the following subsections.

STD-BIOS Data Structures

The data area for the STD-BIOS is in absolute memory locations 00400H through 005FFH, which conforms to the industry standard. Table 2-4 summarizes the assignments within this block of memory. Refer to Appendix B for a detailed description of these data fields.

Table 2-4. STD-BIOS Data Area Summary

Address	Assigned Function
400H-407H	RS-232 Communication Port Addresses
408H-40FH	Parallel Printer Port Addresses
410H-416H	Equipment Flag
417H-43DH	Keyboard Data Area
43Eh-448H	Flexible Disc Data Area
449H-466H	Video Display Data Area
467H-46BH	Option ROM Data Area
46CH-470H	Timer Data Area
471H-473H	System Data Flags
474H-477H	Hard Disc Data Area
478H-47FH	Printer Time out Counters
480H-483H	Keyboard Buffer Pointers
484H-488H	Enhanced Graphics Adapter (EGA) Data Area
489H-48AH	Reserved for Display Adapters
48BH-48BH	Flexible Disc Data Rate Area
48CH-48FH	Extended Hard Disc Data Area
490H-495H	Extended Flexible Disc Data Area
496H-497H	Keyboard Mode Indicator/LED Data Area
498H-4A0H	Real-Time Clock Data Area
4A1H-4A7H	Reserved for Network Adapter Cards
4A8H-4ABH	Pointer to EGA Data Area
4ACH-4EFH	Reserved
4F0H-4FFH	Intra-application Communication Area
500H-500H	Print Screen Status
501H-503H	Reserved
504H-504H	DOS Data Area
505H-5FFH	Reserved

EX-BIOS Data Structures

Data structures for the EX-BIOS drivers are located in a block of memory at the top of system RAM. The address of this block varies depending on the amount of RAM contained in the system and the hardware configuration.

There are three types of data structures in the EX-BIOS data area. These structures are: the HP_VECTOR_TABLE and its associated HP_ENTRY_CODE, the driver data areas, and the EX-BIOS global data area.

The HP_VECTOR_TABLE

Each of the CPU interrupt vectors contains the Code Segment default (CS) and Instruction Pointer (IP) of its associated service routine. The HP_ENTRY interrupt vector (default 06FH) contains the CS:IP of the HP_ENTRY_CODE. This routine uses the value contained in the BP register (an offset into the HP_VECTOR_TABLE, vector address) to branch to the appropriate EX-BIOS driver. The HP_VECTOR_TABLE resides at the base of the EX-BIOS data area. The HP_VECTOR_TABLE consists of an array of 3-word (six bytes) entries, one for each EX-BIOS driver. Each entry consists of the IP, CS, and Data Segment (DS) of a driver.

Figure 2-2 illustrates the relationship between the CPU interrupt vectors, the HP_VECTOR_TABLE, HP_ENTRY_CODE, and the EX-BIOS drivers.

The HP_ENTRY_CODE

The CS:IP in the HP_ENTRY interrupt vector points to a piece of code which branches to the desired EX-BIOS driver. The vector address passed in BP must be a multiple of six. The code is as follows:

```
HP_ENTRY_CODE:
     MOV      DS,CS:[BP+4]
     JMP      FAR PTR CS:[BP]
```

This code resides directly after the last entry in the HP_VECTOR_TABLE. Therefore, the CS:IP entry in the HP_ENTRY interrupt vector provides two further pieces of information. CS:0 is the starting address of the HP_VECTOR_TABLE and IP is the length of the HP_VECTOR_TABLE.

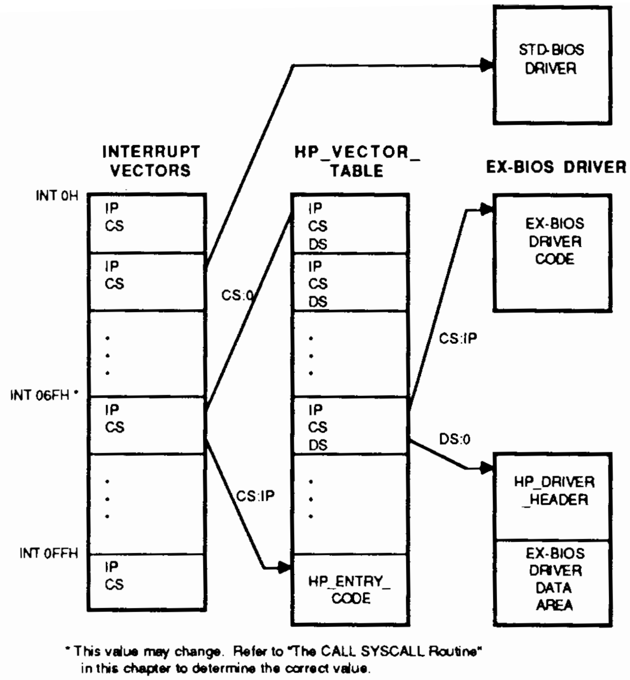


Figure 2-2. Interrupt Vectors and HP_VECTOR_TABLE

Driver Data Areas

Each driver has an independently specified data area. Some EX-BIOS drivers share the same data areas. The data areas for the EX-BIOS drivers are above the HP_VECTOR_TABLE and the HP_ENTRY_CODE shown in Figure 2-2. Although each driver has its own data area, the DS for each driver is stored in the HP_VECTOR_TABLE, and its data area must start at DS:0. Each data area must reside on a paragraph boundary.

The data area for each driver consists of a driver header, followed by an optional variable storage area. The variable storage area is unique to each driver. Table 2-5 provides a general description of the contents of an EX-BIOS driver header.

Table 2-5. HP_DRIVER_HEADER

Variable	Offset Description	Offset	Type
DH_ATR	Driver Attribute Field	0	Word
DH_NAME_INDEX	Driver String Index Field	2	Word
DH_V_DEFAULT	Driver's Default Logical Device Vector	4	Word
DH_P_CLASS	Driver's Parent Class	6	Word
DH_C_CLASS	Driver's Child Class	8	Word
DH_V_PARENT	Driver's Parent Vector	0AH	Word
DH_V_CHILD	Driver's Child Vector	0CH	Word
DH_MAJOR	Sub Address Field	0EH	Byte
DH_MINOR	Sub Address Field	0FH	Byte

EX-BIOS Driver Headers

The following defines each of the EX-BIOS driver header fields. Additional information on these fields can be found in Appendix G.

DH_ATR:	Each bit in the DH_ATR field indicates a property of the driver for device mapping purposes. These bits are defined in Appendix G.
DH_NAME_INDEX:	The DH_NAME_INDEX is used to derive the localization string index of the driver. This string index is given by the function F_STR_GET_STRING in the V_SYSTEM driver. See Chapter 8 for additional information.
DH_V_DEFAULT:	The DH_V_DEFAULT field contains the driver's default vector address.
DH_P_CLASS and DH_C_CLASS:	In conjunction, these fields indicate which drivers may be mapped together. DH_P_CLASS and DH_C_CLASS are bit masks. Each bit position represents a set of drivers. If a bit is set, then the driver is in that set of drivers. The DH_P_CLASS field indicates a driver is in from 0 to 16 different driver sets. A driver can only map to another driver if its DH_P_CLASS field matches at least one bit position of the other driver's DH_C_CLASS field. Furthermore, the DH_ATR field is another condition of mapping. The bits are defined in Appendix G.
DH_V_PARENT:	The DH_V_PARENT field contains a vector to the driver that is called when the current driver receives an F_ISR function code that it cannot or doesn't know how to process.
DH_V_CHILD:	The DH_V_CHILD field contains a vector to the driver that is called if this driver decides it cannot handle the request function (as long as that function is not F_ISR).
DH_MAJOR and DH_MINOR:	Device bus address information.

EX-BIOS Global Data Area

The method for locating the EX-BIOS global data area is found in the "EX-BIOS Data Area Map" of Appendix B. The EX-BIOS global data area is shared between several EX-BIOS drivers. It contains temporary and permanent variables that are required by the BIOS to function properly. Some of these variables can be modified by application programs. As with any modification to the STD-BIOS data area, care should be taken with the EX-BIOS global data area. Table 2-6 defines the contents of this area.

Table 2-6. Definition of Global Data Area Contents

Byte	Type	Name of Driver	Definition
0-1DH		Reserved	
1EH	Word	T_STR_NEXT_INDEX	Next unused string index number.
20H and up		Reserved	

3

Video

The HP Multimode Video Display Adapter provides a wide variety of display modes, resolution, character attributes, and other features. The purpose of the video driver is to allow programs to access these features and control the video display.

Overview

In the text mode, the Multimode Video Display Adapter uses an 8 x 16 character cell which generates high quality characters. Access to the display memory is fully synchronized to eliminate the "snow" problem present in many color display adapters. (Snow occurs when writing a character to display memory while the video memory is being accessed by the display refresh circuitry.) This full synchronization makes the INT 10H video driver faster, since there is no need to wait for a vertical retrace to place characters on the screen.

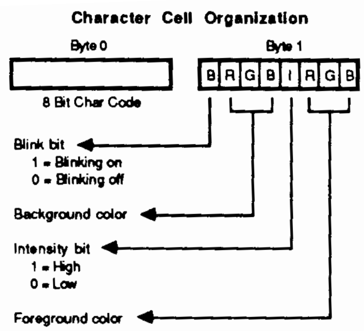
The Multimode Video Display Adapter provides seven more display modes than the industry standard color graphics adapter. Four of the modes allow 27 lines of text on the screen. The other three modes allow graphics modes that double the graphics resolution of the display (320x400 and 640x400 pixels). The standard INT 10H video driver has been extended to allow the programmer to set these modes. No other support is provided to make use of these modes. Refer to *HP Vectra Accessories Technical Reference Manual* (for either the Vectra ES or RS) for more information on the Multimode Video Display Adapter.

Data Structures

The Multimode Video Display Adapter has 32KB of video memory starting at address 0B8000H. This allows graphics resolutions of 320x400 in medium resolution modes and 640x400 in high resolution modes. The following is a discussion of how this memory is organized, depending on the video mode selected.

In either of the text modes (80x25 or 40x25), memory is organized as sequential pages. Each page contains character cells that are made up of an 8-bit character code and an 8-bit attribute (see Figure 3-1).

Graphics modes can be of two types: medium resolution (320x200 or 320x400) and high resolution (640x200 or 640x400). In the medium resolution mode, each pixel corresponds to two bits of memory, so four colors can be displayed. In the high resolution modes, each pixel corresponds to one bit of memory, and only one color can be displayed (the background color is always black). See Figures 3-2 and 3-3 for more details.



Color Values

I	R	G	B	Color
0	0	0	0	Black
0	0	0	1	Blue
0	0	1	0	Green
0	0	1	1	Cyan
0	1	0	0	Red
0	1	0	1	Magenta
0	1	1	0	Brown
0	1	1	1	Light Grey
1	0	0	0	Dark Grey
1	0	0	1	Light Blue
1	0	1	0	Light Green
1	0	1	1	Light Cyan
1	1	0	0	Light Red
1	1	0	1	Light Magenta
1	1	1	0	Yellow
1	1	1	1	White

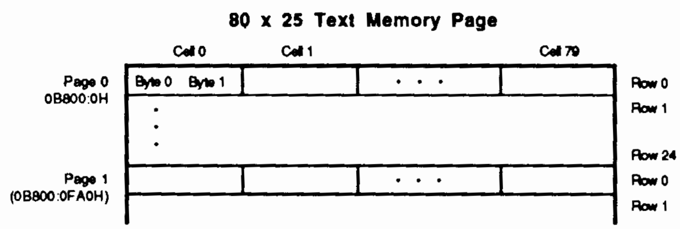
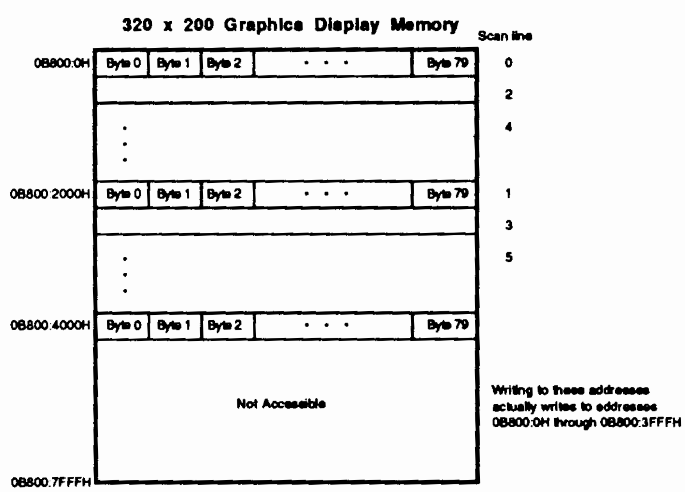


Figure 3-1. Text Display Memory Organization



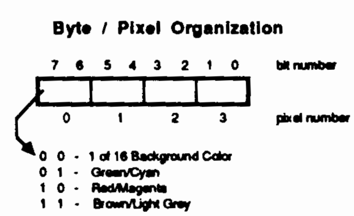


Figure 3-2. 320 x 200 Graphics Display Memory Organization

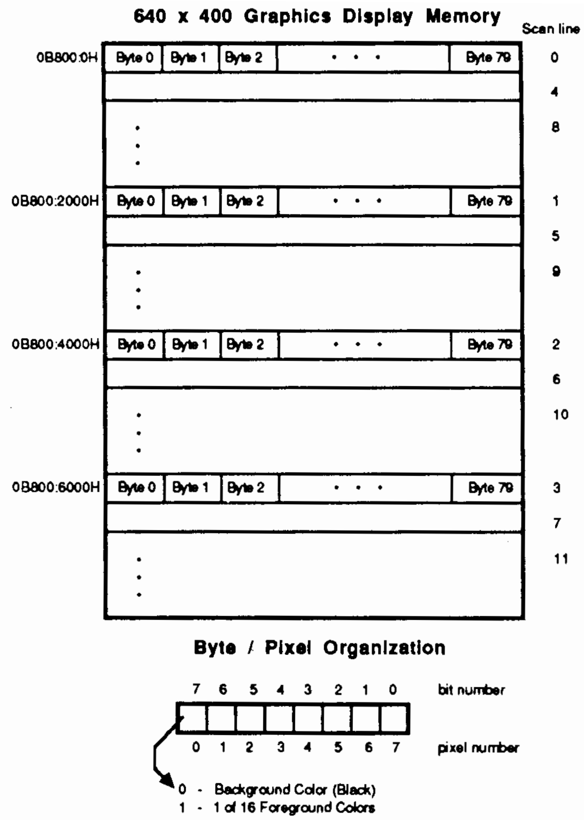


Figure 3-3. 640 x 400 Graphics Display Memory Organization

In all the graphics modes, the memory used for scan lines is not sequential but it is interleaved at fixed intervals of 8K. In the modes that are 200 scan lines, even scan lines start at offset 0 and odd scan lines start at offset 2000H. In the modes that are 400 scan lines, the following table can be used to determine the appropriate offset:

```
Scan line is multiple of 4
             (0,4,8,12 ...) use offset 0
Scan line is multiple of 4 plus 1
             (1,5,9,13 ...) use offset 2000H
Scan line is multiple of 4 plus 2
             (2,6,10,14...) use offset 4000H
Scan line is multiple of 4 plus 3
             (3,7,11,15...) use offset 6000H
```

All the scan lines of a particular group are organized sequentially within a particular offset. See Figures 3-2 and 3-3.

Other video driver data structures are located in the STD-BIOS data area. They are stored in memory addresses 449H (40H:49H) through 466H (40H:66H). Table 3-1 lists the STD-BIOS Video Driver memory locations and their definitions.

Table 3-1. STD-BIOS Video Driver Data Area

Address	Type	Definition
00449H	Byte	Current Video Display Mode
0044AH	Word	Number of columns
0044CH	Word	Regen Buffer length
0044EH	Word	Starting address of regen buffer
00450H	Word	Cursor position for Display Page 0
00452H	Word	Cursor position for Display Page 1
00454H	Word	Cursor position for Display Page 2
00456H	Word	Cursor position for Display Page 3
00458H	Word	Cursor position for Display Page 4
0045AH	Word	Cursor position for Display Page 5
0045CH	Word	Cursor position for Display Page 6
0045EH	Word	Cursor position for Display Page 7
00460H	Word	Current cursor mode
00462H	Byte	Active page number
00463H	Word	Address of current display adapter
00465H	Byte	Mode (current setting of status register)
00466H	Byte	Pallet setting

Video data structures are also maintained in the EX-BIOS data area. These structures are accessible through the data segment of the EX-BIOS video service routine. The following code sets the ES register to the EX-BIOS video driver's (V_SVIDEO'S) data segment:

NOTE

The current value of HP_ENTRY must be determined once using the "CALL SYSCALL" routine.

```
MOV AX,0                     ;segment at 0
MOV ES,AX                    ;
MOV AX,ES: [HP_ENTRY*4+2]    ;read the base address
                             ;of the HP_VECTOR_TABLE
MOV ES,AX
MOV AX,ES: [V_SVIDEO+4]      ;read base address of (V_SVIDEO = 54H)
MOV ES,AX                    ;video parameters
```

The addresses listed are offsets into this data segment. Table 3-2 gives the data maintained in V_SVIDEO's (0054H) data segment:

Table 3-2. Video EX-BIOS Data Structures

Variable Name	Definition	Offset	Type
Driver Header	Device Header Attributes, Name, Index, and Default Vector	0-5	Byte
VID_PRIMARY	The current primary display: 00 - Card at I/O Address 3B0H 01 - Card at I/O Address 3C0H 02 - Card at I/O Address 3D0H 03 - Card containing ROM Code.	6	Byte
VID_SECONDARY	If two cards are in the system, same number as VID_PRIMARY for the second card.	7	Byte
VID_FOUND_ROM	Flag set to true if ROM code is found in any video adapter card.	8	Byte
VID_IDS	of IDs of all cards found.	9-0CH	Byte
VID_STATUS	RAM copies of the status register.	0D-010H	Byte
VID_EXT_STATUS	RAM copies of the extended status register for each possible card in the system.	11-014H	Byte
VID_PARM_BLOCK	Reserved for saving the video parameters stored in the standard BIOS data area when switching between primary and secondary video boards.	15-03BH	Byte
VID_LAST _IBM_MODE	Used to detect if a 'rogue' program changed the modes without telling the HP system.	03CH	Byte
VID_EXT_MODE	Specifies the current video mode (0 . . . 15).	03DH	Byte
	Reserved	03E-03F	Byte

Video Driver (INT 10H)

The video driver functions (summarized in Table 3-3) can be broken down into the following categories.

- Display Control--These functions control the display appearance, cursor and light pen position, active text memory page, and scrolling through text memory.
- Character Handling Functions--These functions manipulate characters on the screen.
- String Functions--These functions allow placement of strings of text on the screen.
- Graphics Functions--These functions provide an interface to the graphics capabilities of the machine.
- Extended Video Functions--These functions support extra video capabilities of the Multimode Video Display Adapter hardware.

Table 3-3. Video Driver Function Code Summary

Equate	Value	Definition
00H	F10_SET_MODE	Set video mode
01H	F10_SET_CURSIZE	Set cursor size
02H	F10_SET_CURPOS	Set cursor position
03H	F10_RD_CURPOS	Read cursor position
04H	F10_RD_PENPOS	Read light pen position
05H	F10_SET_PAGE	Set active display page
06H	F10_SCROLL_UP	Scroll rectangle up
07H	F10_SCROLL_DN	Scroll rectangle down
08H	F10_RD_CHARATR	Read character and attribute at cursor position
09H	F10_WR_CHARATR	Write character and attribute at cursor position
0AH	F10_WR_CHARCUR	Write character at cursor position
0BH	F10_SET_PALLET	Set color pallet
0CH	F10_WR_PIXEL	Write pixel
0DH	F10_RD_PIXEL	Read pixel
0EH	F10_WR_CHARTEL	Write teletype character
0FH	F10_GET_STMODE	Get video state and mode
10H-12H		Reserved
	Write string functions:	
1300H	F10_WRS_00	Global attribute
1301H	F10_WRS_01	Global attribute, move cursor
1302H	F10_WRS_02	Individual attributes
1303H	F10_WRS_03	Individual attributes, move cursor
	Extented video functions:	
6F00H	F10_INQUIRE	EX-BIOS present
6F01H	F10_GET_INFO	Get video parameters
6F02H	F10_SET_INFO	Sets video parameters
6F03H	F10_MOD_INFO	Modifies video parameters
6F04H	F10_GET_RES	Reports video resolution
6F05H	F10_XSET_MODE	Sets video resolution

Video Driver Function Definitions

The following gives a detailed description of each of the functions in the video driver.

F10_SET_MODE (AH = 00H)

This function sets the display mode of the video adapter. The new mode is determined by the value passed in the AL register. Refer to the *Vectra Accessories Technical Reference Manual* (for either the Vectra ES or RS) for additional information on the various video display modes available on the Multimode Video Display Adapter.


```
On Entry: AH = F10_SET_MODE (00H)
          AL = Mode

               Data Definition

               00  40 x 25  Black and White Alphanumeric
               01  40 x 25  Color Alphanumeric
               02  80 x 25  Black and White Alphanumeric
               03  80 x 25  Color Alphanumeric
               04 320 x 200 Color Graphics
               05 320 x 200 Black and White Graphics
               06 640 x 200 Black and White Graphics
               07 Only valid if a monochrome display adapter is present.

On Exit: No values returned

Registers Altered: AX
```

F10_SET _CURSIZE (AH = 01H)

This function sets the size of the cursor displayed in the alphanumeric display modes. Each character cell in the alphanumeric display modes is eight scan lines high. The cursor size is defined by specifying the starting and ending scan lines within the character cell. The scan lines are numbered from 0 (top of cell) to 7 (bottom). The starting and ending scan lines are passed in registers CH and CL. This function performs no operation if the Multimode Video Display Adapter is in one of the graphics modes.

```
On Entry: AH = F10_SET_CURSIZE (01H)
          CH = Starting scan line
          CL = Ending scan line

On Exit: No values returned.

Registers Altered: AH
```

F10_SET _CURPOS (AH = 02H)

This function sets the row and column address of the cursor to the specified page and moves the cursor to that address. When the Multimode Video Display Adapter is in one of the graphics modes, a page number of 0 must be specified.

```
On Entry: AH = F10_SET_CURPOS (02H)
          BH = Display page number
          DH = Row address of cursor. (0. . .24)
          DL = Column address of cursor. (0. . .79)

On Exit: No values returned.

Registers Altered: None
```

F10_RD_CURPOS (AH = 03H)

This function returns the current address and size of the cursor on the specified page. If the Multimode Video Display Adapter is in one of the graphics modes, a page number of 0 must be specified. Otherwise, the values returned for the cursor size in the graphics mode will be invalid.

```
On Entry: AH = F10_RD_CURPOS (03H)
          BH = Display page number

On Exit:  CH = Starting scan line
          CL = Ending scan line
          DH = Row address of cursor. (0. . .24)
          DL = Column address of cursor. (0. . .79)

Registers Altered: CX, DX
```

F10_RD_PENPOS (AH = 04H)

This function returns the current state and position of the light pen if it is activated. The position is reported in both character row/column and graphic pixel formats.

```
On Entry: AH = F10_RD_PENPOS (04H)

On Exit:  AH = Light Pen state

               Data Definition:

               0  Not activated
               1  Activated

          BX = Horizontal pixel position of light pen
          CH = Vertical pixel position of light pen (200 line mode)
          DH = Row position of light pen
          DL = Column position of light pen

Registers Altered: AH, BX, CH, DX
```

F10_SET_PAGE (AH = 05H)

This function sets the active display page in the alphanumeric mode. Valid page numbers are 0 through 7 for 80 x 25 modes, and 0 through 7 for 40 x 25 modes. This function is not valid for graphics modes.

```
On Entry: AH = F10_SET_PAGE (05H)
          AL = Page number (0 through 7)
```

On Exit: No values returned.

Registers Altered: AX

F10_SCROLL_UP (AH = 06H)

This function scrolls the contents of a window up a specified number of lines. The window is defined by the row and column addresses stored in the CX and DX registers. The number of lines to be scrolled is passed in register AL. If AL is set to 0, the function interprets this as a command to scroll all lines.

```
On Entry: AH = F10_SCROLL_UP (06H)
          AL = Number of lines to scroll (0 = scroll all)
          BH = Attribute to place in blanked lines
          CH = Row address of upper left corner of window (0. . .24)
          CL = Column address of upper left corner of window (0. . .79)
          DH = Row address of lower right corner of window (0. . .24)
          DL = Column address of lower right corner of window (0. . .79)
```

On Exit: No values returned.

Registers Altered: None

F10_SCROLL_DN (AH = 07H)

This function scrolls the contents of a window down a specified number of lines. The window is defined by the row and column addresses stored in the CX and DX registers. The number of lines to be scrolled is passed in register AL. If AL is set to 0, the function interprets this as a command to scroll all lines. This function is only valid when the Multimode Video Display Adapter is in one of the alphanumeric modes.

```
On Entry: AH = F10_SCROLL_DN (07H)
          AL = Number of lines to scroll (0 = scroll all)
          BH = Attribute to place in blanked lines
          CH = Row address of upper left corner of window (0. . .24)
          CL = Column address of upper left corner of window (0. . .79)
          DH = Row address of lower right corner of window (0. . .24)
          DL = Column address of lower right corner of window (0. . .79)
```

On Exit: No values returned.

Registers Altered: None

F10_RD_CHARATR (AH = 08H)

This function returns the character byte and attribute byte at the current cursor location. If the Multimode Video Display Adapter is in one of the alphanumeric modes, a page number must be specified. If the video display adapter is in one of the graphics modes, only the character is returned, since characters do not have attribute bytes in the graphics modes.

```
On Entry: AH = F10_RD_CHARATR (08H)
          BH = Page number (alphanumeric modes only)

On Exit:  AH = Attribute byte (valid only in alphanumeric modes)
          AL = Character

Registers Altered: AX
```

F10_WR_CHARATR (AH = 09H)

This function writes character and attribute bytes at the current cursor location. If the Multimode Video Display Adapter is in one of the alphanumeric modes, a page number may be specified. If the Multimode Video Display Adapter is in one of the graphics modes, only the character is written. More than one character and attribute can be stored by placing the number of copies desired in CX. This function will wrap around both line and screen if too many characters are specified. Note that this function makes copies of a single character/attribute combination; it does not print a string. Refer to the Write String function for that operation.

```
On Entry: AH = F10_WR_CHARATR (09H)
          AL = Character
          BH = Page number (alphanumeric modes only)
          BL = Attribute byte (valid only in alphanumeric modes)
          CX = Number of characters to write

On Exit: No values returned.

Registers Altered: None
```

F10_WR_CHARCUR (AH = 0AH)

This function writes a character to the current cursor location, retaining the existing attribute byte. The function is identical to the F10_WR_CHARATR function, except that no attribute byte is written.

```
On Entry: AH = F10_WR_CHARCUR (0AH)
          AL = Character
          BH = Page number (alphanumeric modes only)
          CX = Number of characters to write

On Exit: No values returned.

Registers Altered: None
```

F10_SET_PALLET (AH = OBH)

This function allows setting the background color (if BH = 0) or the foreground color pallet (if BH = 1).

```
On Entry: AH = F10_SET_PALLET (OBH)
          BH = Color Select ID

               Data Definition

                0   Set the background color (in medium resolution
                    modes) or the foreground color (in high resolution
                    modes) based on the low bits of BL (bits 0. . .3)
                    to one of 16 colors.

                1   Select color pallet (for medium resolution modes)
                    based on the least significant bit of BL. If
                    bit 0 of BL = 0 then select the green, red,
                    brown pallet. If bit 0 of BL = 1 then select the
                    cyan, magenta, light gray pallet.

          BL = Color select value

On Exit: No values returned

Registers Altered: None
```

F10_WR_PIXEL (AH = OCH)

This function writes a pixel on the screen. If the Multimode Video Display Adapter is in one of the "Four color" modes (320 x 200) the color of the pixel may be passed in register AL. Bits 0 and 1 of AL are interpreted as the color bits. If bit 7 of AL is set, bits 0 and 1 are "XORed" with the current pixel color bits, otherwise they replace the current pixel color bits. If the Multimode Video Display Adapter is in the "Two color" mode (640 x 200), the bit corresponding to the desired pixel is set.

```
On Entry: AH = F10_WR_PIXEL (OCH)
          AL = Color

          In "Four color" mode (320x200):

          Bit   Data   Definition

           7     1     Bits 0 and 1 XORed with
                       current pixel.
                 0     Bits 0 and 1 replace current pixel.
          0,1          Color bits.
```

In "Two color" mode (640x200):

Bit	Data	Definition
7	1	Bit 0 XORed with current pixel.
	0	Bit 0 replaces current pixel.
0		Color bit.

CX = Horizontal pixel address
DX = Vertical pixel address

On Exit: No values returned.

Registers Altered: AX

F10_RD_PIXEL (AH = 0DH)

This function returns the color code of the specified pixel.

On Entry: AH = F10_RD_PIXEL (0DH)
CX = Horizontal pixel address
DX = Vertical pixel address

On Exit: AL = Color value of pixel

Registers Altered: AX, CX, DX

F10_WR_CHARTEL (AH = 0EH)

This function writes a character to the active page, then advances the cursor one location. At the end of a line, the cursor will wrap to the next line; at the end of the screen, the cursor will scroll. In the alphanumeric modes, this function maintains the current video display attributes. In the graphics modes, the foreground color is passed in register BL. The ASCII characters Line Feed (0AH), Carriage Return (0DH), Backspace (08H), Bell (07H), and Tab (09H) are interpreted by this function as ASCII commands and are executed as such.

On Entry: AH = F10_WR_CHARTEL (0EH)
AL = Character
BL = Foreground color (in graphics modes only)

On Exit: No values returned.

Registers Altered: AX

F10_GET _STMODE (AH = 0FH)

This function returns the current Multimode Video Display Adapter state. The mode, number of characters per line, and current display page are returned.

```
On Entry: AH = F10_GET_STMODE (0FH)

On Exit:  AH = Number of characters per line
          AL = Current mode
          BH = Current display page

Registers Altered: AX, BH
```

Write String (AH = 13H)

This function writes a string of characters to the screen. This function consists of four separate subfunctions which control whether each character has its own attribute byte or not, and whether the cursor is moved or not. Each of the subfunctions is detailed in the following. The ASCII characters Line Feed (0AH), Carriage Return (0DH), Backspace (08H), Bell (07H), and Tab (09H) are interpreted by this function as ASCII commands and are executed as such.

F10_WRS_00 (AX = 1300H)

Write string attribute without moving cursor.

```
On Entry: AX = F10_WRS_00 (1300H)
          BH = Display page number
          BL = String attribute byte
          CX = Length of string
          DH = Row address of first character
          DL = Column address of first character
       ES:BP = Pointer to start of string.  Format of string is:
               Char, Char, . . ., Char

On Exit: No values returned.

Registers Altered: None
```

F10_WRS_01 (AX = 1301H)

Write string attribute and move cursor.

```
On Entry: AX = F10_WRS_01 (1301H)
          BH = Display page number
          BL = String attribute byte
          CX = Length of string
          DH = Row address of first character
          DL = Column address of first character
       ES:BP = Pointer to start of string.  Format of string is:
               Char, Char, . . ., Char

On Exit: No values returned.

Registers Altered: None
```

F10_WRS_02 (AX = 1302H)

Write character attribute without moving cursor.

```
On Entry: AX = F10_WRS_02 (1302H)
          BH = Display page number
          CX = Length of string
          DH = Row address of first character
          DL = Column address of first character
       ES:BP = Pointer to start of string.  Format of string is:
               Char, Attr, Char, Attr, ..., Char, Attr

On Exit: No values returned.

Registers Altered: None
```

F10_WRS_03 (AX = 1303H)

Write character attribute and move cursor.

```
On Entry: AX = F10_WRS_03 (1303H)
          BH = Display page number
          CX = Length of string
          DH = Row address of first character
          DL = Column address of first character
       ES:BP = Pointer to start of string.  Format of string is:
               Char, Attr, Char, Attr, ..., Char, Attr

On Exit: No values returned.

Registers Altered: None
```

HP Extended Video Functions

This set of functions support the features of the Multimode Video Display Adapter which are not covered using the standard video functions. This function consists of separate subfunctions which support the various extended capabilities of the Multimode Video Display Adapter (implemented through the EX-BIOS). Each of these subfunctions is defined in the following subsections.

F10_INQUIRE (AX = 6F00H)

This subfunction determines whether or not the extended HP functions are available. If the extended video functions are available, the BX register will be set to 4850H (which is the ASCII characters "HP").

```
On Entry: AX = F10_INQUIRE (6F00H)
          BX = Any value except 4850H ('HP')

On Exit:  BX = 'HP' (4850H)

Registers Altered: AX, BX
```

F10_GET_INFO (AX = 6F01H)

This function returns information about the active display adapter.

```
On Entry: AX = F10_GET_INFO (6F01H)

On Exit:  AH = Status register information

               Bit Data Definition

                0   1   Display Enabled.
                1   1   Light Pen Trigger Set.
                2   1   Light Pen Switch Made.
                3   1   Vertical Sync.
                4       Monitor Resolution
                    0   350 or 400 line monitor
                    1   200 line monitor
                5       Display type
                    0   Color
                    1   Monochrome
               6-7      Diagnostic Bits

          AL = Card Identifier
```

Data	Definition
00H	Non HP card with ROM and possibly its own INT 10H driver.
41H	Multimode Video Display Adapter
42H	Reserved
43H	Reserved
44H	Reserved
45H	Industry Standard Monochrome Display Adapter
46H	Industry Standard Color Display Adapter
51H	Reserved

CL = Current value of Extended Control register. This register is only valid when the Card Identifier is 41H.

This description applies to data returned when a Multimode Video Display Adapter is in the system.

Bit	Data	Definition
0		Current screen resolution
	0	200 line
	1	400 line
1		Underline enable.
	0	Blue bit of foreground attribute interpreted as color blue.
	1	Blue bit of foreground attribute interpreted as underline.
2		Font Selected.
	0	PC-8
	1	HP ROMAN8
3		Memory disable.
	0	Memory enabled for CPU access.
	1	Memory disabled for CPU access.
4		16/32K Memory select.
	0	Wrap second 16K of RAM into first 16K.
	1	Allow access to full 32K of memory.
5		Page select.
	0	Select first 16K of memory.
	1	Select second 16K of memory.
6-7		Unused

Registers Altered: AX, CL

F10_SET_INFO (AX = 6F02H)

This function modifies the value of the Extended Control register port 3DDH on the Multimode Video Display Adapter. (Refer to the *Vectra Accessories Technical Reference Manual* - for either the Vectra ES or RS - for more information about this port.)

```
On Entry: AX = F10_SET_INFO (6F02H)
          BL = Byte of data to be written to
               the Extended Control Register.

               Bit Data Definition

                0       Current screen resolution
                     0  200 line
                     1  400 line
                1       Underline enable.
                     0  Blue bit of foreground attribute
                        interpreted as color blue.
                     1  Blue bit of foreground attribute
                        interpreted as underline.
                2       Font Selected.
                     0  PC-8
                     1  HP ROMAN8
                3       Memory disable.
                     0  Memory enabled for CPU access.
                     1  Memory disabled for CPU access.
                4       16/32K Memory select.
                     0  Wrap second 16K of RAM into first 16K.
                     1  Allow access to full 32K of memory.
                5       Page select.
                     0  Select first 16K of memory.
                     1  Select second 16K of memory.
               6-7      Reserved

On Exit: No values returned.

Registers Altered: AX, BL
```

F10_MOD_INFO (AX = 6F03H)

This function modifies individual bits in the Extension Control register (port 3DDH) of the Multimode Video Display Adapter. A mask byte is passed in register BH, which allows individual bits to be modified without changing the state of other mode bits in the register.

```
On Entry: AX = F10_MOD_INFO (6F03H)
          BH = Mask. Bits with a mask value of "1" are not modified; bits
               with a mask value of "0" are modified.
          BL = Bits to change. The bits indicated by the mask (BH)
               take the value of the BL register.

               Bit Data Definition

                0       Current screen resolution
                     0  200 line
                     1  400 line
                1       Underline enable.
                     0  Blue bit of foreground attribute
                        interpreted as color blue.
                     1  Blue bit of foreground attribute
                        interpreted as underline.
                2       Font Selected.
                     0  PC-8
                     1  HP ROMAN8
                3       Memory disable.
                     0  Memory enabled for CPU access.
                     1  Memory disabled for CPU access.
                4       16/32K Memory select.
                     0  Wrap second 16K of RAM into first 16K.
                     1  Allow access to full 32K of memory.
                5       Page select.
                     0  Select first 16K of memory.
                     1  Select second 16K of memory.
               6-7      Reserved
```

On Exit: No values returned.

Registers Altered: AX

Example:

```
MOV AX,F10_MOD_INFO    ; EX-BIOS Function - Modify Ex-Reg (6F03H)
MOV BL,00000100B       ; Select Character Font: HP ROMAN8
MOV BH,11111011B       ; Only Modify Character Font
INT 10H                ; Call Video Interrupt
```

F10_GET_RES (AX = 6F04H)

This function returns the current video mode and screen resolution.

```
On Entry: AX = F10_GET_RES (6F04H)

On Exit:  AL = Current video mode (See Set Mode.)

               Data Definition

               00H  40 x 25 Alphanumeric Black and White
               01H  40 x 25 Alphanumeric Color
               02H  80 x 25 Alphanumeric Black and White
               03H  80 x 25 Alphanumeric Color
               04H 320 x 200 Graphics Color
               05H 320 x 200 Graphics Black and White
               06H 640 x 200 Graphics Black and White
               07H  80 x 25 Only Valid for Monochrome Cards
               08H  80 x 27 Alphanumeric Black and White
               09H  80 x 27 Alphanumeric Color
               0AH  40 x 27 Alphanumeric Black and White
               0BH  40 x 27 Alphanumeric Color
               0CH 640 x 400 2 Color
               0DH 640 x 400 Graphics Black and White
               0EH 320 x 400 Graphics Color
               0FH 320 x 400 Graphics Black and White
```

If in one of the graphics modes:

```
          BX = Horizontal resolution in pixels
          CX = Vertical resolution in pixels
```

If in one of the text modes:

```
          BX = Number of characters per row
          CX = Number of rows
```

Registers Altered: AX, BX, CX

F10_XSET_MODE (AX = 6F05H)

This function places the HP Multimode Video Display Adapter in one of sixteen possible modes of operation. Modes 0 through 7 are identical to the modes available with function F10_SET_MODE of the video driver. Modes 8 through 15 are unique to the HP Vectra and the Multimode Video Display Adapter and may only be set using this function.

Programmers must exercise caution when setting video modes with both F10_SET_MODE (0H) and F10_XSET_MODE (6F05H). Whenever F10_XSET_MODE is used to select one of the "HP only" modes (8-15), F10_XSET_MODE (not F10_SET_MODE) must be used to return to one of the industry standard modes (0-7). This "pairing" of function calls is necessary because F10_XSET_MODE modifies an I/O port not normally affected by the industry standard modes. F10_SET_MODE does not deal with this I/O port.

```
On Entry: AX = F10_XSET_MODE (6F05H)
          BL = Video mode

               Data Definition

               00H  40 x 25 Alphanumeric Black and White
               01H  40 x 25 Alphanumeric Color
               02H  80 x 25 Alphanumeric Black and White
               03H  80 x 25 Alphanumeric Color
               04H 320 x 200 Graphics Color
               05H 320 x 200 Graphics Black and White
               06H 640 x 200 Graphics Black and White
               07H  80 x 25 Only Valid for Monochrome Cards
               08H  80 x 27 Alphanumeric Black and White
               09H  80 x 27 Alphanumeric Color
               0AH  40 x 27 Alphanumeric Black and White
               0BH  40 x 27 Alphanumeric Color
               0CH           Reserved
               0DH 640 x 400 Graphics Black and White
               0EH 320 x 400 Graphics Color
               0FH 320 x 400 Graphics Black and White

On Exit: No values returned.

Altered Registers: AX, BL
```

Example:

```
MOV AX,F10_XSET_MODE     ; Call EX-BIOS function Set mode (6F05H)
MOV BL,0DH               ; Select 640 x 400 line mode
INT INT_VIDEO            ; Call video interrupt (10H)
```

4

Input System and HP-HIL

The Input System is a set of drivers which support the HP-HIL input devices. Up to seven HP-HIL input devices may be connected at one time. The Input System can support properly integrated non-HP-HIL devices as well.

Overview

The standard devices that connect to the system via the HP-HIL link are the mouse, touchscreen, and tablet. The interfaces for simple mouse, touchscreen, and tablet support are described in this Chapter.

The architecture of the Input System is divided into two levels (see Figure 4-1). The application interface level allows the programmer to communicate with the HP-HIL devices with minimum overhead. The second level, the hardware interface level, allows programmers to manipulate the internals of the system. With this interface, support for additional devices can be added or the data path of existing ones re-directed.

The first portion of this chapter provides an overview of the application interface level, a detailed description of the actual interfaces, and how to access them. The second portion of this chapter describes the hardware interface level.

Application Interface Level

Application programs interface with the Input System through a set of logical device drivers. The Input System has an application interface for the tablet, pointer (simple mouse), and touchscreen input devices. The Input System device drivers are shown in Figure 4-1.

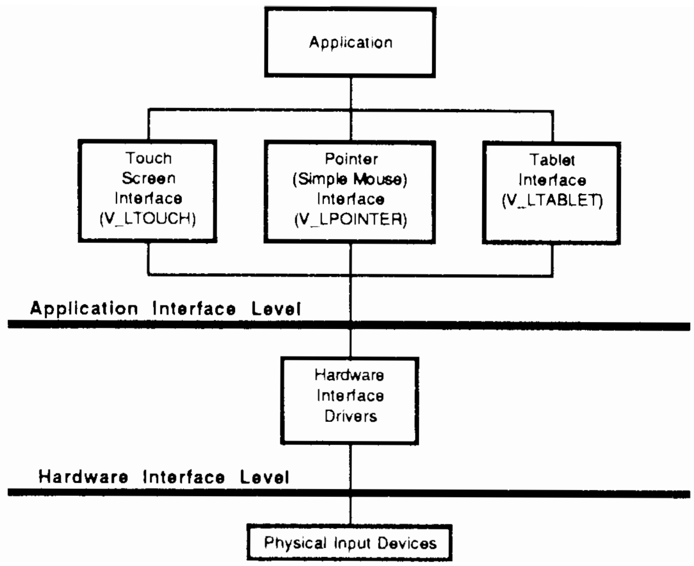


Figure 4-1. Input System Block Diagram

The tablet, pointer, and touchscreen application program interface drivers are grouped together in Figure 4-1 as they are all Graphic Input Device (GID) drivers. GID drivers accept relative graphic motion data, absolute graphics data, and button scancode data from the input devices. Data from these devices is represented in a consistent manner throughout the Input System, making programmatic access to different Graphic Input Devices a simple task (see the Application Event Driver Example later in this chapter).

Overview

The Input System supports three logical GID drivers; one for each of the standard GID data types. There is a GID driver for each of the touchscreen, pointer (simple mouse), and tablet devices called V_LTOUCH, V_LPOINTER, and V_LTABLET respectively. Each of these drivers has a fixed location in the HP_VECTOR_TABLE. They all share a common code module (i.e., they have the same CS:IP in the table), but have different data areas.

The GID drivers perform clipping and scaling under certain conditions. Absolute devices like the touchscreen and tablet are always scaled but clipping is user selectable. Relative devices like the mouse can have both scaling and clipping selected by the user.

The logical GID drivers perform two additional tasks. The first is graphics cursor movement (sprite tracking). This is performed by the EX-BIOS driver V_STRACK, which is called by the logical GID driver if tracking is enabled. The second task is to provide interrupt service to the application. The application may install a routine to be called by the logical GID driver every time a GID event occurs, as opposed to the application calling the GID driver repeatedly (polling) to see if an event has occurred.

The following text outlines the actions that occur for touchscreen input, from touching the screen to application data retrieval.

1. The user touches the screen. This causes the physical device to generate input data and interrupt the hardware interface level.

2. The hardware interface level processes the interrupt and passes the data (ISR Event Record) to the logical touchscreen driver (V_LTOUCH).

3. V_LTOUCH scales the event to fit the current dimensions of the screen. At this point two optional things may happen. First, the data may be clipped. Second, the user defined event driver will be called if it is installed and enabled.

4. If the user event routine was not installed and enabled, then the application must call (poll) V_LTOUCH with the F_SAMPLE function (see V_LTOUCH functions) to get the input data.

There are two methods for applications to receive data from the Input System: polled mode and interrupt mode. In polled mode, the application must continually interrogate the logical GID driver using the F_SAMPLE function to determine if any input has occurred. In interrupt mode, the application must first install an ISR event handling routine (application event driver) using SF_CREATE_EVENT to handle interrupt calls from the logical GID driver. After installation, the application informs the logical GID driver that it is ready to receive interrupts by calling the SF_EVENT_ON subfunction. After event interrupts have been enabled, the application will receive an interrupt every time the logical GID driver receives data from the hardware interface level.

Data Structures

The application interface level uses two major data structures: the Logical Describe Record and the Logical ISR Event Record(s). These data structures help keep track of the numerous events occurring in the Input System.

Logical Describe Record

The Logical Describe Record is used by the logical GID drivers to keep track of the current state of their respective devices. Each of the logical GID drivers has a Logical Describe Record associated with it, which is located directly after the driver header starting with memory address DS:0010H. Table 4-1 lists the field types and offsets of the Logical GID Driver Describe Record. An explanation of the Logical GID Driver Describe Record follows the table.

Table 4-1. Logical GID Driver Describe Record

Field	Description	Type	Offset
Driver Header	Driver Header (see Chapter 2)		00H
LD_SOURCE	Device GID type	BYTE	10H
LD_HPHIL_ID	Physical device ID	BYTE	11H
LD_DEVICE_STATE	Status bits for the logical device	WORD	12H
LD_INDEX	Physical device vector number	BYTE	14H
LD_MAX_AXIS	Maximum number of axes reported	BYTE	15H

Table 4-1. Logical GID Driver Describe Record (Cont.)

Field	Description	Type	Offset
LD_CLASS	Device class	BYTE	16H
LD_PROMPTS	Number of button/prompts	BYTE	17H
LD_PARAGRAPHS	Size of this record in paragraphs	BYTE	18H
LD_RESERVED	Reserved	BYTE	19H-1BH
LD_TRANSITION	Button transitions	BYTE	1CH
LD_STATE	Current state of the buttons	BYTE	1DH
LD_RESOLUTION	Logical device resolution	WORD	1EH
LD_SIZE_X	Maximum x-axis count	WORD	20H
LD_SIZE_Y	Maximum y-axis count	WORD	22H
LD_ABS_X	X position data for absolute devices	WORD	24H
LD_ABS_Y	Y position data for absolute devices	WORD	26H
LD_REL_X	X delta for relative devices	WORD	28H
LD_REL_Y	Y delta for relative devices	WORD	2AH
LD_ACCUM_X	X-axis scaling accumulator	WORD	2CH
LD_ACCUM_Y	Y-axis scaling accumulator	WORD	2EH
LD_SIZE_Z	Maximum z-axis count	WORD	30H
LD_ABS_Z	Z position data for absolute devices	WORD	32H
LD_REL_Z	Z delta for relative devices	WORD	34H
LD_ACCUM_Z	Z-axis scaling accumulator	WORD	36H

Logical Describe Record Definitions

LD_SOURCE — This field is divided into nibbles. Bits 7-4 contain the graphics input device type. This field is loaded with the low order nibble of the appropriate logical GID data type (Table 4-5). Bits 3-0 are reserved.

LD_HPHIL_ID — ID byte of the physical device which last reported data. Table 4-2 lists the HP-HIL device ID bytes.

Table 4-2. HP-HIL Device ID Bytes

Device Type	ID Range	Device Description
Other	00H-2BH	Reserved
	2CH-2FH	Tone Generator
	30H-3FH	Reserved
Character Entry	40H-4FH	Reserved
	50H-5BH	Reserved
	5CH-5FH	Bar code Reader
Relative Positioners	60H-67H	Reserved
	68H-6BH	Mouse
	6CH-6FH	Trackball
	70H-7FH	Reserved
Absolute Positioners	80H-87H	Reserved
	88H-8BH	Touchpad
	8CH-8FH	Touchscreen
	90H-97H	Graphics Tablet
	98H-9FH	Reserved
Other	0A0H-0FFH	Reserved

LD_DEVICE_STATE — Status bits for the logical device:

Bit	Definition
0FH-05H	Reserved.
04H	Event enabled when set.
03H	Tracking enabled when set.
02H	Clipping enabled when set.
01H	Button error occurred when set.
00H	Interrupt in progress when set.

LD_INDEX — This contains the vector address divided by 6 of the last physical device that reported data.

LD_MAX_AXIS — Maximum number of axes supported by the device. Valid range is 0-2.

LD_CLASS	Device class. Bits 7-4 contain the current class. Bits 3-0 contain the default class. See Appendix G for more information on device classes.
LD_PROMPTS	Number of buttons and prompts supported by the device. Bits 7-4 contain the number of prompts. Bits 3-0 contain the number of buttons.
LD_PARAGRAPHS	Size of this record in paragraphs: 0 means 3 paragraphs, 1 means 4 paragraphs.
LD_TRANSITION	Transitions reported per button, i.e., a set bit indicates that the corresponding button was either pushed or released. Bit 7 corresponds to button 7 etc.
LD_STATE	Current state of the buttons. 1 is down, 0 is up. Bit 7 corresponds to button 7 etc. If LD_STATE is XOR'ed with LD_TRANSITION the result is the previous button state.
LD_RESOLUTION	This is the resolution of the logical device. For logical devices this is typically one.
LD_SIZE_X	Maximum count (in units of resolution) for the x-axis.
LD_SIZE_Y	Maximum count (in units of resolution) for the y-axis.
LD_ABS_X	X position data for devices which report absolute coordinates (absolute devices).
LD_ABS_Y	Y position data for devices which report absolute coordinates.
LD_REL_X	Latest change in x position for devices which return coordinates relative to the previous position (relative devices).
LD_REL_Y	Latest change in y position for devices which return coordinates relative to the previous position.
LD_ACCUM_X	Accumulator used to sum partial movements when scaling from the physical device space to the logical device space. The value stored here represents a fraction of one logical unit for the x-axis.
LD_ACCUM_Y	Accumulator used to sum partial movements when scaling from the physical device space to the logical device space. The value stored here represents a fraction of one logical unit for the y-axis.
LD_SIZE_Z	Maximum count (in units of resolution) for the z-axis.
LD_ABS_Z	Z position data for devices which report absolute coordinates.
LD_REL_Z	Latest change in z position for devices which return coordinates relative to the previous position.
LD_ACCUM_Z	Accumulator used to sum partial movements when scaling from the physical device space to the logical device space. The value stored here represents a fraction of one logical unit for the z-axis.

Logical ISR Event Records

A Logical ISR Event Record is not a data structure in the truest sense, but is a set of register definitions for inter-driver communication of input events. These definitions apply not only to Input System drivers but to application event drivers as well. The following define the Logical ISR Event Records.

GID Button ISR Event Record

```
     AH = F_ISR (00H)
     DL = Physical device driver's vector address / 6
     BX = Button information.

          Bit    Value    Definition

       0FH-08H    -       Reserved
       07H        1       Button up
                  0       Button down
       06H-00H    -       Button number (0-7)

     DH = Data Type
   ES:0 = Pointer to Physical device driver
          header and Physical Describe Record.
```

GID Motion ISR Event Record

```
     AH = F_ISR (00H)
     DL = Physical device driver's vector address / 6
     BX = X axis motion in raw data form.
     CX = Y axis motion in raw data form.
     SI = Z axis motion in raw data form.
     DH = Data Type
   ES:0 = Pointer to physical device driver
```

header and Physical Describe Record.

The button number in the Button information field (BX) denotes which button on the device is reporting data. Of special interest is button seven (proximity indicator) which is currently used by absolute devices to indicate that the device measurement field is active. For example, someone is touching the touchscreen, or the stylus is in contact with the tablet surface.

The Data Type field (DH) contains a code representing the current type of logical GID data stored in the event record. For button events this value will be T_KC_BUTTON. For logical GID motion events, permissible types are: T_TS, T_POINTER and T_TABLET, which correspond to data originating from V_LTOUCH, V_LPOINTER, and V_LTABLET respectively. For a complete list of logical GID event data types see Table 4-3.

Table 4-3. Logical GID Event Data Types

Type	Value	Definition
T_KC_BUTTON	09H	Button data
T_TS	45H	Specially formed data (80x25--default) generated by V_LTOUCH
T_TABLET	46H	Specially formed data (640x200 range--default) generated by V_LTABLET
T_POINTER	47H	Specially formed data (640x200 range--default) generated by V_LPOINTER

Application Event Drivers

As previously mentioned, applications may install a routine to handle interrupts from the logical GID drivers. Three predefined vectors in the HP_VECTOR_TABLE are initialized to the null driver (V_PNULL). The three vectors are V_EVENT_TOUCH, V_EVENT_POINTER, and V_EVENT_TABLET which are called by the logical GID drivers V_LTOUCH, V_LPOINTER, and V_LTABLET respectively when event interrupts are enabled by a call to SF_EVENT_ON. A call to SF_CREATE_EVENT sets the corresponding event vector to point to the user application event

The application event driver is required to support only one function, F_ISR. The driver should return RS_UNSUPPORTED for all unimplemented functions.

Logical GID Drivers

The drivers V_LTOUCH, V_LPOINTER, and V_LTABLET represent the application interface to the Input System. These drivers provide functions to poll for data, enable/disable application event interrupts, enable/disable tracking, and enable/disable clipping and/or scaling.

V_LTOUCH Driver (BP = 00C6H)

This section contains a detailed description of the touchscreen driver. Table 4-4 is a summary of the touchscreen driver function code.

Table 4-4. Touchscreen Driver Function Code Summary

Function Value	Function Equate	Definition
	V_LTOUCH	Application interface to Touchscreen
00	F_ISR	Logical Interrupt
02	F_SYSTEM	System functions
02/00	SF_INIT	Initialize the driver data area
02/02	SF_START	Start driver
02/04	SF_REPORT_STATE	Report state of device
02/06	SF_VERSION_DESC	Report driver version number
02/08	SF_DEF_ATTR	Set default logical scaling attributes
02/0A	SF_GET_ATTR	Get scaling attributes
02/0C	SF_SET_ATTR	Set scaling attributes
04	F_IO_CONTROL	I/O Control functions
04/00	SF_LOCK	Unsupported
04/02	SF_UNLOCK	Unsupported
04/04	SF_TRACK_ON	Turn cursor track on
04/06	SF_TRACK_OFF	Turn cursor track off
04/08	SF_CREATE_EVENT	Establish a new routine to be called on logical device events
04/0A	SF_EVENT_ON	Enable event call to parent driver
04/0C	SF_EVENT_OFF	Disable event call to parent driver
04/0E	SF_CLIPPING_ON	Enable logical device clipping
04/10	SF_CLIPPING_OFF	Disable logical device clipping
06	F_SAMPLE	Report absolute position of GID

Touchscreen Driver Functions Definitions

F_ISR (AH = 00H)

This function receives an ISR Event record from one of the physical GID drivers. The calling driver has handled the physical interrupt and updated the Physical Describe Record to reflect the event. This function translates the physical event into the logical coordinate system and calls its parent, V_EVENT_TOUCH, (if EVENT is enabled). In addition, this function passes the event to V_STRACK so that the sprite can be updated (if TRACK is enabled). This function is a response to a logical hardware interrupt and not user callable.

```
On Entry: AH = F_ISR (00H)
          DH = Data Type
          DL = Physical device driver's vector index.
        ES:0 = Pointer to Physical device driver header and Physical
               Describe Record.
          BP = V_LTOUCH (00C6H)

          For Button Event:
          BX = Button information.

          Bit       Value  Definition

          0FH-08H     -    Reserved
          07H         1    Button up
                      0    Button down
          06H-00H     -    Button number (0-7)

          For Motion Event:
          BX = X axis motion in raw data form.
          CX = Y axis motion in raw data form.

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

Related Functions:  SF_CREATE_EVENT, SF_EVENT_ON, SF_TRACK_ON
```

SF_INIT (AX = 0200H)

This subfunction is called to initialize the driver. Refer to Chapter 8 for a complete discussion of the protocol used in data space allocation.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BX = "Last used DS" in HP Data Area
          BP = V_LTOUCH (00C6H)

On Exit: AH = Return Status Code
         BX = New "last used DS" in HP Data Area

Registers Altered: AX, BX, BP, DS
```

SF_START (AX = 0202H)

This subfunction starts the logical touchscreen driver.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_START (02H)
          BP = V_LTOUCH (00C6H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_REPORT_STATE (AX = 0204H)

This subfunction returns the LD_DEVICE_STATE field from the Logical Describe Record.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_REPORT_STATE (04H)
          BP = V_LTOUCH (00C6H)

On Exit: AH = Return Status Code
         DX = LD_DEVICE_STATE from Logical Describe Record

Registers Altered: AX, DX, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_LTOUCH (00C6H)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

SF_DEF_ATTR (AX = 0208H)

This subfunction sets the attributes of the logical touchscreen driver to their default values. The default attributes for the touch screen driver are: LD_SIZE_X = 79 and LD_SIZE_Y = 24.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_DEF_ATTR (08H)
          BP = V_LTOUCH (00C6H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_GET_ATTR (AX = 020AH)

This subfunction returns the current scaling attributes, LD_SIZE_X and LD_SIZE_Y.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_GET_ATTR (0AH)
          BP = V_LTOUCH (00C6H)

On Exit: AH = Return Status Code
         BX = LD_SIZE_X (logical size along X axis)
         CX = LD_SIZE_Y (logical size along Y axis)

Registers Altered: AX, BX, CX, BP, DS
```

SF_SET_ATTR (AX = 020CH)

This subfunction sets the scaling attributes, LD_SIZE_X, and LD_SIZE_Y, in the Logical Describe Record.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_SET_ATTR (0CH)
          BX = LD_SIZE_X (logical size along X axis)
          CX = LD_SIZE_Y (logical size along Y axis)
          BP = V_LTOUCH (00C6H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_TRACK_ON (AX = 0404H)

This subfunction turns tracking on. For each movement of the logical device, V_STRACK will be called to update the graphics cursor (sprite) position.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_TRACK_ON (04H)
          BP = V_LTOUCH (00C6H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF__TRACK__OFF (AX = 0406H)

This subfunction turns tracking off.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_TRACK_OFF (06H)
          BP = V_LTOUCH (00C6H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF__CREATE__EVENT (AX = 0408H)

This subfunction establishes the routine to be called on logical device events. The IP, CS, and DS of the routine are passed to this subfunction. These values are exchanged with the vector entry of the V__EVENT__TOUCH driver in the HP__VECTOR__TABLE, V__EVENT__TOUCH being the parent of the logical touchscreen driver. The IP, CS, and DS of the previous routine are returned to the caller. Note that this subfunction does not enable the event call to the parent routine; this must be done explicitly using SF__EVENT__ON.

The ISR event records passed to the V__EVENT__TOUCH driver will have one of the following two formats, depending on the Data Type stored in DL.

```
V_EVENT_TOUCH Button ISR Event Record:

          AH = F_ISR (00H)
          DL = Physical device drivers vector address / 6
          BX = Button information.

          Bit       Value  Definition

          0FH-08H     -    Reserved
          07H         1    Button up
          06H-00H     -    Button number (0-7)

          DH = Data Type
        ES:0 = Pointer to V_LTOUCH device driver
               header and Logical Describe Record.

V_EVENT_TOUCH Motion ISR Event Record:

          AH = F_ISR (00H)
          DL = Physical device driver's vector address / 6
          BX = A number between 0 and LD_SIZE_X
          CX = A number between 0 and LD_SIZE_Y
          DH = Data Type
        ES:0 = Pointer to V_LTOUCH device driver
               header and Logical Describe Record.
```

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CREATE_EVENT (08H)
          BP = V_LTOUCH (00C6H)
          DX = DS of new V_EVENT_TOUCH routine
          SI = IP of new V_EVENT_TOUCH routine
          ES = CS of new V_EVENT_TOUCH routine

On Exit: AH = Return Status Code
         DX = DS of previous V_EVENT_TOUCH routine
         SI = IP of previous V_EVENT_TOUCH routine
         ES = CS of previous V_EVENT_TOUCH routine

Registers Altered: AX, DX, SI, BP, ES, DS

Related Functions: SF_EVENT_ON
```

The following example shows how to use the SF_CREATE_EVENT function. The routine EVENT will be the event procedure that is called when events are enabled.

```
EVENT  PROC FAR
     CMP  AH,F_ISR ;only support function F_ISR
     JE   PROCESS_EVENT
     MOV  AH, RS_UNSUPPORTED
     IRET
PROCESS_EVENT:
     .                              ; code to process data
     .                              ; (see touchscreen
     .                              ; event record)
     MOV  AH, RS_SUCCESSFUL         ; return successful completion
     IRET
EVENT ENDP

     MOV  AH, F_IO_CONTROL
     MOV  AL, SF_CREATE_EVENT
     MOV  BP, V_LTOUCH
     MOV  DX, DS        ; want to use the current data segment for event DS
     PUSH CS
     POP  ES            ; current CS also segment of event routine
     LEA  SI,CS:EVENT ; get the IP of the event routine
     PUSH DS            ; save current DS
     CALL SYSCALL       ; call extended BIOS driver
     POP  DS
```

SF_EVENT_ON (AX = 040AH)

This subfunction enables the event (parent) call to the touchscreen event routine (V_EVENT_TOUCH). The link to the touchscreen event routine must have already been established using SF_CREATE_EVENT.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_EVENT_ON (0AH)
          BP = V_LTOUCH (00C6H)
```

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

Related Functions: SF_CREATE_EVENT, SF_EVENT_OFF

SF_EVENT_OFF (AX = 040CH)

This subfunction disables the call to the touchscreen event routine.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_EVENT_OFF (0CH)
          BP = V_LTOUCH (00C6H)
```

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

SF_CLIPPING_ON (AX = 040EH)

This subfunction enables logical device clipping. Physical device motion will be scaled to logical space and will be clipped to avoid overflow or underflow. Clipping is activated for both absolute and relative motion.

When there is a relative device mapped to this device driver, clipping works the best. It will make sure that the new position always falls within the logical space.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CLIPPING_ON (0EH)
          BP = V_LTOUCH (00C6H)
```

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

SF_CLIPPING_OFF (AX = 0410H)

This subfunction disables logical device clipping. Physical device motion will be scaled to logical space, but overflow or underflow may occur.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CLIPPING_OFF (10H)
          BP = V_LTOUCH (00C6H)
```

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

F_SAMPLE (AH = 06H)

This function allows an application to poll the touchscreen device. This function reports the current absolute position of the logical device in a form similar to a Logical ISR Event Record.

```
On Entry: AH = F_SAMPLE (06H)
          BP = V_LTOUCH (00C6H)

On Exit: AH = Return Status Code
         BX = Current logical position along X axis
         CX = Current logical position along Y axis
         DL = LD_TRANSITION field of Logical Describe Record
         DH = LD_STATE field of Logical Describe Record
       ES:0 = Pointer to logical device header and Describe Record

Registers Altered: AX, BX, CX, DX, BP, DS, ES
```

The following is an example of how to call the F_SAMPLE function.

```
     MOV  AH, F_SAMPLE    ; load function code
     MOV  BP, V_LTOUCH    ; load vector address
     PUSH DS              ; save the current DS
     CALL SYSCALL         ; call extended BIOS driver
     POP  DS              ; restore DS
```

V_LPOINTER Driver (BP = 00C0H)

This section contains a detailed description of the pointer driver. Table 4-5 summarizes the functions supported by the pointer driver.

Table 4-5. Pointer Driver Function Code Summary

Function Equate	Definition	Vector Address	Func. Value
V_LPOINTER	Application interface to Pointer/Mouse	00C0H	
F_ISR	Logical Interrupt	00C0H	00
F_SYSTEM	System functions	00C0H	02
SF_INIT	Initialize the driver data area	00C0H	02/00
SF_START	Start driver	00C0H	02/02
SF_REPORT _STATE	Report state of device	00C0H	02/04
SF_VERSION _DESC	Report driver version number	00C0H	02/06
SF_DEF _ATTR	Set default logical scaling attributes	00C0H	02/08
SF_GET _ATTR	Get scaling attributes	00C0H	02/0A
SF_SET _ATTR	Set scaling attributes	00C0H	02/0C
F_IO_CONTROL	I/O Control Functions	00C0H	04
SF_LOCK	Unsupported	00C0H	04/00
SF_UNLOCK	Unsupported	00C0H	04/02

Table 4-5. Pointer Driver Function Code Summary (Cont.)

Function Equate	Definition	Vector Address	Func. Value
SF_TRACK_ON	Turn cursor track on	00C0H	04/04
SF_TRACK_OFF	Turn cursor track off	00C0H	04/06
SF_CREATE_EVENT	Establish a new routine to be called on logical device events	00C0H	04/08
SF_EVENT_ON	Enable event call to parent driver	00C0H	04/0A
SF_EVENT_OFF	Disable event call to parent driver	00C0H	04/0C
SF_CLIPPING_ON	Enable logical device clipping	00C0H	04/0E
SF_CLIPPING_OFF	Disable logical device clipping	00C0H	04/10
F_SAMPLE	Report absolute position of GID	00C0H	06

Pointer Driver Function Definitions

F_ISR (AH = 00H)

This function receives an ISR Event record from one of the physical GID drivers. The calling driver has handled the physical interrupt and updated the Physical Describe Record to reflect the event. This function translates the physical event into the logical coordinate system and calls its parent, V_EVENT_POINTER, (if EVENT is enabled). In addition, this function passes the event to V_STRACK so that the sprite can be updated (if TRACK is enabled). This function is a response to a logical hardware interrupt and not user callable.

```
On Entry: AH = F_ISR (00H)
          DH = Data Type
          DL = Physical device drivers vector index.
        ES:0 = Pointer to physical device driver
               header and Physical Describe Record.
          BP = V_LPOINTER (00C0H)

          For Button Event:
          BX = Button information.

          Bit       Value     Definition

          0FH-08H   -         Reserved
          07H       1         Button up
                    0         Button down
          06H-00H   -         Button number (0-7)

          For Motion Event:
          BX = X axis motion in raw data form.
          CX = Y axis motion in raw data form.
          SI = Z axis motion in raw data form.
```

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

Related Functions: SF_CREATE_EVENT, SF_EVENT_ON, SF_TRACK_ON

SF_INIT (AX = 0200H)

This subfunction is called to initialize the driver. Refer to Chapter 8 for a complete discussion of the protocol used in data space allocation.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BX = "Last used DS" in HP Data Area
          BP = V_LPOINTER (00C0H)

On Exit: AH = Return Status Code
         BX = New "last used DS" in HP Data Area
```

Registers Altered: AX, BX, BP, DS

SF_START (AX = 0202H)

This subfunction starts the logical pointer driver.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_START (02H)
          BP = V_LPOINTER (00C0H)

On Exit: AH = Return Status Code
```

Registers Altered: AX, BP, DS

SF_REPORT_STATE (AX = 0204H)

This subfunction returns the LD_DEVICE_STATE field from the Logical Describe Record.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_REPORT_STATE (04H)
          BP = V_LPOINTER (00C0H)

On Exit: AH = Return Status Code
         DX = LD_DEVICE_STATE from Logical Describe Record
```

Registers Altered: AX, DX, BP, DS

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_LPOINTER (00C0H)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

SF_DEF_ATTR (AX = 0208H)

This subfunction sets the attributes of the logical pointer driver to their default values. The default attributes for the pointer driver are: LD_SIZE_X = 639, LD_SIZE_Y = 199 and LD_SIZE_Z = 100.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_DEF_ATTR (08H)
          BP = V_LPOINTER (00C0H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_GET_ATTR (AX = 020AH)

This subfunction returns the current scaling attributes, LD_SIZE_X, LD_SIZE_Y and LD_SIZE_Z.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_GET_ATTR (0AH)
          BP = V_LPOINTER (00C0H)

On Exit: AH = Return Status Code
         BX = LD_SIZE_X (logical size along X axis)
         CX = LD_SIZE_Y (logical size along Y axis)
         SI = LD_SIZE_Z (logical size along Z axis)

Registers Altered: AX, BX, CX, BP, DS
```

SF_SET_ATTR (AX = 020CH)

This subfunction sets the scaling attributes, LD_SIZE_X, LD_SIZE_Y and LD_SIZE_Z in the Logical Describe Record.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_SET_ATTR (0CH)
          BX = LD_SIZE_X (logical size along X axis)
          CX = LD_SIZE_Y (logical size along Y axis)
          SI = LD_SIZE_Z (logical size along Z axis)
          BP = V_LPOINTER (00C0H)
```

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

SF_TRACK_ON (AX = 0404H)

This subfunction turns tracking on. For each movement of the logical device, V_STRACK will be called to update the graphics cursor (sprite) position.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_TRACK_ON (04H)
          BP = V_LPOINTER (00C0H)
```

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

SF_TRACK_OFF (AX = 0406H)

This subfunction turns tracking off.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_TRACK_OFF (06H)
          BP = V_LPOINTER (00C0H)
```

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

SF_CREATE_EVENT (AX = 0408H)

This subfunction establishes the routine to be called on logical device events. The IP, CS, and DS of the routine are passed to this subfunction. These values are exchanged with the vector entry of the V_EVENT_POINTER driver in the HP_VECTOR_TABLE, V_EVENT_POINTER being the parent of the logical pointer driver. The IP, CS, and DS of the previous routine are returned to the caller. Note that this subfunction does not enable the event call to the parent routine; this must be done explicitly using SF_EVENT_ON.

The ISR event records passed to the V_EVENT_POINTER driver will have one of the following two formats depending on the Data Type stored in DL.

```
V_EVENT_POINTER Button ISR Event Record:
          AH = F_ISR (00H)
          DL = Physical device driver's vector address / 6
          BX = Button information.

          Bit      Value  Definition

          0FH-08H    -    Reserved
          07H        1    Button up
                     0    Button down
          06H-00H    -    Button number (0-7)

          DH = Data Type
        ES:0 = Pointer to V_LPOINTER device driver
               header and Logical Describe Record.
```

V_EVENT_POINTER Motion ISR Event Record:

```
          AH = F_ISR (00H)
          DL = Physical device driver's vector address / 6
          BX = Relative movement in the X direction
               (Positive number indicates movement to the right)
          CX = Relative movement in the Y direction
               (Positive number indicates movement down)
          DH = Data Type
        ES:0 = Pointer to V_LPOINTER device driver header and
               Logical Describe Record.

On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CREATE_EVENT (08H)
          BP = V_LPOINTER (00C0H)
          DX = DS of new V_EVENT_POINTER routine
          SI = IP of new V_EVENT_POINTER routine
          ES = CS of new V_EVENT_POINTER routine

On Exit: AH = Return Status Code
         DX = DS of previous V_EVENT_POINTER routine
         SI = IP of previous V_EVENT_POINTER routine
         ES = CS of previous V_EVENT_POINTER routine

Registers Altered: AX, DX, SI, BP, ES, DS

Related Functions: SF_EVENT_ON
```

This example shows how to use the SF_CREATE_EVENT function. The routine EVENT will be the event procedure that is called when events are enabled.

```
EVENT  PROC FAR
     CMP  AH, F_ISR            ; only support function F_ISR
     JE   PROCESS_EVENT
     MOV  AH, RS_UNSUPPORTED
     IRET
PROCESS_EVENT:
     .                         ; code to process data (see
     .                         ; pointer event record)
     .
     MOV  AH, RS_SUCCESSFUL ; return successful completion
     IRET
EVENT  ENDP

     MOV  AH, F_IO_CONTROL
     MOV  AL, SF_CREATE_EVENT
     MOV  BP, V_LPOINTER
     MOV  DX, DS          ; want to use the current data segment for event DS
     PUSH CS
     POP  ES              ; current CS is also segment of event routine
     LEA  SI, CS:EVENT ; get the IP of the event routine
     PUSH DS              ; save current DS
     CALL SYSCALL         ; call extended BIOS driver
     POP  DS
```

SF_EVENT_ON (AX = 040AH)

This subfunction enables the event (parent) call to the pointer event routine (V_EVENT_POINTER). The link to the pointer event routine must have already been established using SF_CREATE_EVENT.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_EVENT_ON (0AH)
          BP = V_LPOINTER (00C0H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

Related Functions:  SF_CREATE_EVENT, SF_EVENT_OFF
```

SF_EVENT_OFF (AX = 040CH)

This subfunction disables the call to the pointer event routine.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_EVENT_OFF (0CH)
          BP = V_LPOINTER (00C0H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_CLIPPING_ON (AX = 040EH)

This subfunction enables logical device clipping. Physical device motion will be scaled to logical space and will be clipped to avoid overflow or underflow. Clipping is activated for both absolute and relative motion.

When there is a relative device mapped to this device driver, clipping works the best. It will make sure that the new position always falls within the logical space.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CLIPPING_ON (0EH)
          BP = V_LPOINTER (00C0H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_CLIPPING_OFF (AX = 0410H)

This subfunction disables logical device clipping. Physical device motion will be scaled to logical space, but overflow or underflow may occur.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CLIPPING_OFF (10H)
          BP = V_LPOINTER (0C0H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

F_SAMPLE (AH = 06H)

This function allows an application to poll the pointer device. This function reports the current absolute position of the logical device in a form similar to a Logical ISR Event Record.

```
On Entry: AH = F_SAMPLE (06H)
          BP = V_LPOINTER (00C0H)

On Exit: AH = Return Status Code
         BX = Current logical position along X axis
         CX = Current logical position along Y axis
         SI = Current logical position along Z axis
         DL = LD_TRANSITION field of Logical Describe Record
         DH = LD_STATE field of Logical Describe Record
       ES:0 = Pointer to logical device header and Describe Record

Registers Altered: AX, BX, CX, DX, 3P, DS, ES
```

```
MOV AH, F_SAMPLE    ; load function code
MOV BP, V_LPOINTER  ; load vector address
PUSH DS             ; save the current DS
CALL SYSCALL        ; call extended BIOS driver
POP DS              ; restore DS
```

V_LTABLET Driver (BP = 00BAH)

This section contains a detailed description of the tablet driver. See Table 4-6 for a summary of functions supported by the tablet driver.

Table 4-6. Tablet Driver Function Code Summary

Vector Address	Function Value	Function Equate	Definition
00BAH		V_LTABLET	Application interface to Tablet
00BAH	00	F_ISR	Logical Interrupt
00BAH	02	F_SYSTEM	System functions
00BAH	02/00	SF_INIT	Initialize the driver data area
00BAH	02/02	SF_START	Start driver
00BAH	02/04	SF_REPORT _STATE	Report state of device
00BAH	02/06	SF_VERSION _DESC	Report driver version number
00BAH	02/08	SF_DEF _ATTR	Set default logical scaling attributes
00BAH	02/0A	SF_GET _ATTR	Get scaling attributes
00BAH	02/0C	SF_SET _ATTR	Set scaling attributes
00BAH	04	F_IO _CONTROL	I/O Control Functions
00BAH	04/00	F_SF _LOCK	Unsupported
00BAH	04/02	F_SF _UNLOCK	Unsupported

Table 4-6. Tablet Driver Function Code Summary (Cont.)

Vector Address	Function Value	Function Equate	Definition
00BAH	04/04	F_SF_TRACK_ON	Turn cursor track on
00BAH	04/06	F_SF_TRACK_OFF	Turn cursor track off
00BAH	04/08	F_SF_CREATE_EVENT	Establish a new routine to be called on logical device events
00BAH	04/0A	F_SF_EVENT_ON	Enable event call to parent driver
00BAH	04/0C	F_SF_EVENT_OFF	Disable event call to parent driver
00BAH	04/0E	F_SF_CLIPPING_ON	Enable logical device clipping
00BAH	04/10	F_SF_CLIPPING_OFF	Disable logical device clipping
00BAH	06	F_SAMPLE	Report absolute position of GID

Tablet Driver Functions Definition

F_ISR (AH = 00H)

This function receives an ISR Event record from one of the physical GID drivers. The calling driver has handled the physical interrupt and updated the Physical Describe Record to reflect the event. This function translates the physical event into the logical coordinate system and calls its parent, V_EVENT_TABLET, (if EVENT is enabled). In addition, this function passes the event to V_STRACK so that the sprite can be updated (if TRACK is enabled). This function is a response to a logical hardware interrupt and not user callable.

```
On Entry: AH = F_ISR (00H)
          DH = Data Type
          DL = Physical device driver's vector index.
        ES:0 = Pointer to physical device driver header
               and Physical Describe Record.
          BP = V_LTABLET (00BAH)

          For Button Event:
          BX = Button information.
```

Bit	Value	Definition
0FH-08H		Reserved
	1	Button up
	0	Button down
06H-00H		Button number (0-7)

For Motion Event:
BX = X axis motion in raw data form.
CX = Y axis motion in raw data form.
SI = Z axis motion in raw data form.

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

Related Functions: SF_CREATE_EVENT, SF_EVENT_ON, SF_TRACK_ON

SF_INIT (AX = 0200H)

This subfunction is called to initialize the driver. Refer to Chapter 8 for a complete discussion of the protocol used in data space allocation.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BX = "Last used DS" in HP Data Area
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code
         BX = New "last used DS" in HP Data Area
```

Registers Altered: AX, BX, BP, DS

SF_START (AX = 0202H)

This subfunction starts the logical tablet driver.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_START (02H)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code
```

Registers Altered: AX, BP, DS

SF_REPORT_STATE (AX = 0204H)

This subfunction returns the LD_DEVICE_STATE field from the Logical Describe Record.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_REPORT_STATE (04H)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code
         DX = LD_DEVICE_STATE from Logical Describe Record

Registers Altered: AX, DX, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

SF_DEF_ATTR (AX = 0208H)

This subfunction sets the attributes of the logical tablet driver to their default values. The default attributes for the tablet driver are: LD_SIZE_X = 639, LD_SIZE_Y = 199 and LD_SIZE_Z = 100.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_DEF_ATTR (08H)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_GET_ATTR (AX = 020AH)

This subfunction returns the current scaling attributes, LD_SIZE_X, LD_SIZE_Y and LD_SIZE_Z.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_GET_ATTR (0AH)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code
         BX = LD_SIZE_X (logical size along X axis)
         CX = LD_SIZE_Y (logical size along Y axis)
         SI = LD_SIZE_Z (logical size along Z axis)

Registers Altered: AX, BX, CX, BP, DS
```

SF_SET_ATTR (AX = 020CH)

This subfunction sets the scaling attributes, LD_SIZE_X, LD_SIZE_Y and LD_SIZE_Z in the Logical Describe Record.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_SET_ATTR (0CH)
          BX = LD_SIZE_X (logical size along X axis)
          CX = LD_SIZE_Y (logical size along Y axis)
          SI = LD_SIZE_Z (logical size along Z axis)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_TRACK_ON (AX = 0404H)

This subfunction turns tracking on. For each movement of the logical device, V_STRACK will be called to update the graphics cursor (sprite) location.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_TRACK_ON (04H)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF__TRACK__OFF (AX = 0406H)

This subfunction turns tracking off.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_TRACK_OFF (06H)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF__CREATE __EVENT (AX = 0408H)

This subfunction establishes the routine to be called on logical device events. The IP, CS, and DS of the routine are passed to this subfunction. These values are exchanged with the vector entry of the V__EVENT__TABLET driver in the HP__VECTOR__TABLE, V__EVENT__TABLET being the parent of the logical tablet driver. The IP, CS, and DS of the previous routine are returned to the caller. Note that this subfunction does not enable the event call to the parent routine; this must be done explicitly using SF__EVENT__ON.

The ISR event records passed to the V__EVENT__TABLET driver will have one of the following two formats depending on the data type stored in DL.

Format 1:

```
V_EVENT_TABLET Button ISR Event Record:

          AH = F_ISR (00H)
          DL = Physical device driver's vector address / 6
          BX = Button information.

               Bit      Value Definition

               0FH-08H    -   Reserved
               07H        1   Button up
                          0   Button down
               06H-00H    -   Button number(0-7)

          DH = Data Type
        ES:0 = Pointer to V_LTABLET device driver header
               and Logical Describe Record.
```

Format 2:

```
V_EVENT_TABLET Motion ISR Event Record:

          AH = F_ISR (00H)
          DL = Physical device driver's vector address / 6
          BX = A number between 0 and LD_SIZE_X
          CX = A number between 0 and LD_SIZE_Y
          SI = A number between 0 and LD_SIZE_Z
          DH = Data Type
        ES:0 = Pointer to V_TABLET device driver header and Logical
               Describe Record.

On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CREATE_EVENT (08H)
          BP = V_LTABLET (00BAH)
          DX = DS of new V_EVENT_TABLET routine
          SI = IP of new V_EVENT_TABLET routine
          ES = CS of new V_EVENT_TABLET routine

On Exit: AH = Return Status Code
         DX = DS of previous V_EVENT_TABLET routine
         SI = IP of previous V_EVENT_TABLET routine
         ES = CS of previous V_EVENT_TABLET routine

Registers Altered: AX, DX, SI, BP, ES, DS

Related Functions: SF_EVENT_ON
```

This example shows how to use the SF_CREATE_EVENT function. The routine EVENT will be the event procedure that is called when events are enabled.

```
EVENT   PROC FAR
          CMP  AH, F_ISR          ; only support function F_ISR
          JE   PROCESS_EVENT
          MOV  AH, RS_UNSUPPORTED
          IRET
PROCESS_EVENT:
          .                       ; code to process data (see
          .                       ; tablet event record)
          .
          MOV  AH, RS_SUCCESSFUL  ; return successful completion
          IRET
EVENT   ENDP

          MOV  AH, F_IO_CONTROL
          MOV  AL, SF_CREATE_EVENT
          MOV  BP, V_LTABLET
          MOV  DX, DS             ; want to use the current data segment
                                  ; segment for event DS
          PUSH  CS
          POP  ES                 ; current CS is also segment of event routine
          LEA  SI, CS:EVENT       ; get the IP of the event routine
          PUSH  DS                ; save current DS
          CALL SYSCALL            ; call extended BIOS driver
          POP  DS
```

SF_EVENT_ON (AX = 040AH)

This subfunction enables the event (parent) call to the tablet event routine (V_EVENT_TABLET). The link to the tablet event routine must have already been established using SF_CREATE_EVENT.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_EVENT_ON (0AH)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS

Related Functions:  SF_CREATE_EVENT, SF_EVENT_OFF
```

SF_EVENT_OFF (AX = 040CH)

This subfunction disables the call to the tablet event routine.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_EVENT_OFF (0CH)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_CLIPPING _ON (AX = 040EH)

This subfunction enables logical device clipping. Physical device motion will be scaled to logical space and will be clipped to avoid overflow or underflow. Clipping is activated for both absolute and relative motion.

When there is a relative device mapped to this device driver, clipping works the best. It will make sure that the new position always falls within the logical space.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CLIPPING_ON (0EH)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_CLIPPING_OFF (AX = 0410H)

This subfunction disables logical device clipping. Physical device motion will be scaled to logical space, but overflow or underflow may occur.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CLIPPING_OFF (10H)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

F_SAMPLE (AH = 06H)

This function allows an application to poll the tablet device. This function reports the current absolute position of the logical device in a form similar to a Logical ISR Event Record.

```
On Entry: AH = F_SAMPLE (06H)
          BP = V_LTABLET (00BAH)

On Exit: AH = Return Status Code
         BX = Current logical position along X axis
         CX = Current logical position along Y axis
         SI = Current logical position along Z axis
         DL = LD_TRANSITION field of Logical Describe Record
         DH = LD_STATE field of Logical Describe Record
       ES:0 = Pointer to logical device header and Describe Record

Registers Altered: AX, BX, CX, DX, BP, DS, ES
```

The following is an example of how to call the F_SAMPLE function.

```
PUSH BP, V_LTABLET
MOV AH, F_SAMPLE   ; load function code
MOV BP, V_LTABLET  ; load vector address
PUSH DS            ; save the current DS
CALL SYSCALL       ; call extended BIOS driver
POP DS             ; restore DS
```

Application Event Driver Example

The following program is an example of how to input touchscreen data using application event interrupts. The program installs an application event driver using the SF_CREATE_EVENT function and enables event interrupts using the SF_EVENT_ON function. The event handler supports only the F_ISR function which processes both button and motion Logical ISR Event Records.

NOTE

Since the HP interrupt number can change, all "int HP_ENTRY" lines in the following example should be replaced with "CALL SYSCALL" (this routine finds and uses the current HP interrupt number).

Touch Example

```
 286c
page    59,132
title   TOUCH Example
;***DRIVER HEADER*************************************************
;
;  NAME          TOUCH Example
;
;  DESCRIPTION   This program demonstrates how touch works
;
;  LIST OF SECTIONS
;
;
;*****************************************************************
```

```
                           page
                           HP_SHEADER          struc
0000  0000                 DH_ATR              dw      0
0002  0000                 DH_NAME_INDEX       dw      0
0004  0000                 DH_V_DEFAULT        dw      0
0006  0000                 DH_P_CLASS          dw      0
0008  0000                 DH_C_CLASS          dw      0
000A  0000                 DH_V_PARENT         dw      0
000C  0000                 DH_V_CHILD          dw      0
000E  00                   DH_MAJOR            db      0
000F  00                   DH_MINOR            db      0
0010                       HP_SHEADER          ends
= 006F                     HP_ENTRY            equ     06FH
                           SYSCALL             macro   vector
                            ifnb               <vector>
                                               mov     bp,vector
                            endif
                                               int     HP_ENTRY
                                               endm
= 8000                     ATR_HP              equ     8000H
= 0000                     CL_NULL             equ     0000H
= 0000                     F_ISR               equ     0000H
= 0004                     F_IO_CONTROL        equ     0004H
= 0008                     SF_CREATE_EVENT     equ     0008H
= 000C                     SF_EVENT_OFF        equ     000CH
= 000A                     SF_EVENT_ON         equ     000AH
= 0000                     RS_SUCCESSFUL       equ     0000H
= 0002                     RS_UNSUPPORTED      equ     0002H
= 0009                     T_KC_BUTTON         equ     09H      ; reported by the physical driver to the logical drive
                                                                ; PGID translates T_KC_ITF to T_KC_BUTTON and filters
                                                                ; any other scancode out of the data stream
= 0045                     T_TS                equ     45H      ;  Specially formed data  ( 0, 80 x 0  25 range - defa
= 0006                     V_DOLITTLE          equ     0006H
= 00C6                     V_LTOUCH            equ     00C6H
= 0060                     V_EVENT_TOUCH       equ     0060H
= 0001                     READ_CHAR_ECHO      equ     01H
= 0080                     MAKE_BREAK_BIT      equ     10000000B
= 004C                     TERMINATE_PROC      equ     4CH

0000                       TS_EVENT_HEADR      segment
=                          EXAM_HP_ATTR        equ     ATR_HP
0000  8000                          HP_SHEADER         <EXAM_HP_ATTR,V_EVENT_TOUCH/6,V_EVENT_TOUCH,CL_NULL,CL_NULL,V_
                           TLE,V_DOLITTLE>
0002  0010
0004  0060
0006  0000
0008  0000
000A  0006
000C  0006
000E  00
000F  00

0010                       TS_EVENT_HEADR  ends
0000                       DATA_SEG        segment
```

Touch Example (cont.)

```
????                     SAVE_CS          dw       ?
????                     SAVE_IP          dw       ?
????                     SAVE_DS          dw       ?
  50 [                   STACK            dw       80 dup (?)
         ????
                ]

????                     STK_TOP          dw       ?
                         DATA_SEG         ends
                         CODE_SEG         segment
                                          assume   cs:CODE_SEG,ds:DATA_SEG,ss:DATA_SEG
B8 ---- R                BEGIN:           mov      ax,DATA_SEG            ;Load up the ds register with the data segment
8E D8                                     mov      ds,ax
8E D0                                     mov      ss,ax                  ;The stack segment is also in the code segment
8B 26 00A6 R                              mov      sp,STK_TOP             ;Point to the top of the stack
E8 001D R                                 call     TOUCH_ENABLE
B4 01                    INPUT_LOOP:      mov      ah,READ_CHAR_ECHO      ;Read a character w/echo until "^"
CD 21                                     int      21H
3C 5E                                     cmp      al,"^"                 ;Is this the exit character?
75 F8                                     jne      INPUT_LOOP
E8 0084 R                EXIT_PROG:       call     TOUCH_RESTORE
B4 4C                                     mov      ah,TERMINATE_PROC      ;Exit
CD 21                                     int      21H
                         TOUCH_ENABLE     proc
B4 04                                     mov      ah,F_IO_CONTROL        ;Move my touch event handler into the HP vector tab
                         le
B0 08                                     mov      al,SF_CREATE_EVENT
8C CB                                     mov      bx,cs
8E C3                                     mov      es,bx
8D 36 0048 R                              lea      si,TOUCH_HANDLER
BA ---- R                                 mov      dx,TS_EVENT_HEADR
                                          syscall  V_LTOUCH
BD 00C6                +                           mov      bp,V_LTOUCH
CD 6F                  +                           int      HP_ENTRY
8C C0                                     mov      ax,es                  ;Save the old event values
A3 0000 R                                 mov      word ptr SAVE_CS,ax
89 36 0002 R                              mov      word ptr SAVE_IP,si
89 16 0004 R                              mov      word ptr SAVE_DS,dx
B4 04                                     mov      ah,F_IO_CONTROL        ;Start accepting calls
B0 0A                                     mov      al,SF_EVENT_ON
                                          syscall  V_LTOUCH
BD 00C6                +                           mov      bp,V_LTOUCH
CD 6F                  +                           int      HP_ENTRY
C3                                        ret
                         TOUCH_ENABLE     endp

                         TOUCH_HANDLER    proc
80 FC 00                                  cmp      ah,F_ISR               ;Logical interrupt?
74 03                                     je       PROCESS_ISR            ; yes, continue
B4 02                                     mov      ah,RS_UNSUPPORTED      ;set return code
CF                                        iret
60                       PROCESS_ISR:     pusha                           ;Save all the registers
80 FE 45                                  cmp      dh,T_TS                ;Is this a position report or a make/break report
74 07                                     je       short POS_REPORT
80 FE 09                                  cmp      dh,T_KC_BUTTON
```

Touch Example (cont.)

```
0059  74 0E                                  je      short BUTTON_REPORT
005B  EB 23                                  jmp     short EXIT_TOUCH
005D  B4 02              POS_REPORT:         mov     ah,02H                  ;Move the cursor to the recieved position
005F  8A F1                                  mov     dh,cl                   ;using the standard IBM BIOS int 10
0061  8A D3                                  mov     dl,bl
0063  B7 00                                  mov     bh,0
0065  CD 10                                  int     10H
0067  EB 17                                  jmp     short EXIT_TOUCH        ;That finishes that ISR
0069  F6 C3 80          BUTTON_REPORT:      test    bl,MAKE_BREAK_BIT       ;See if this is a touch or a release
006C  74 0A                                  jz      short BUTTON_PUSH
006E  B5 0E                                  mov     ch,0EH                  ;On a release make the cursor back into
0070  B1 0F                                  mov     cl,0FH                  ;a line.
0072  B4 01                                  mov     ah,1
0074  CD 10                                  int     10H
0076  EB 08                                  jmp     short EXIT_TOUCH        ;That finishes a release ISR
0078  B5 00              BUTTON_PUSH:        mov     ch,0                    ;Make the cursor into a box on touch
007A  B1 0F                                  mov     cl,0fh
007C  B4 01                                  mov     ah,1
007E  CD 10                                  int     10H
0080  61                 EXIT_TOUCH:         popa                            ;Restore all the registers.
0081  B4 00                                  mov     ah,RS_SUCCESSFUL        ;Set the return status
0083  CF                                     iret                            ;Return from the ISR
0084                     TOUCH_HANDLER       endp
0084                     TOUCH_RESTORE       proc
0084  B4 04                                  mov     ah,F_IO_CONTROL         ;Stop accepting calls
0086  B0 0C                                  mov     al,SF_EVENT_OFF
                                             syscall V_LTOUCH
0088  BD 00C6          +                     mov     bp,V_LTOUCH
008B  CD 6F            +                     int     HP_ENTRY
008D  B4 04                                  mov     ah,F_IO_CONTROL         ;Restore the old event handler
008F  B0 08                                  mov     al,SF_CREATE_EVENT
0091  8B 1E 0000 R                           mov     bx,word ptr SAVE_CS
0095  8E C3                                  mov     es,bx
0097  8D 36 0002 R                           lea     si,word ptr SAVE_IP
009B  8B 16 0004 R                           mov     dx,word ptr SAVE_DS
                                             syscall V_LTOUCH
009F  BD 00C6          +                     mov     bp,V_LTOUCH
00A2  CD 6F            +                     int     HP_ENTRY
00A4  C3                                     ret
00A5                     TOUCH_RESTORE       endp
00A5                     CODE_SEG            ends
                                             end     BEGIN
```

Touch Example (cont.)

```
Macros:

                N a m e                 Length

SYSCALL . . . . . . . . . . . . .       0002

Structures and records:

                N a m e                 Width   # fields
                                        Shift   Width   Mask    Initial

HP_SHEADER . . . . . . . . . . .        0010    0009
  DH_ATR . . . . . . . . . . . .        0000
  DH_NAME_INDEX. . . . . . . . .        0002
  DH_V_DEFAULT . . . . . . . . .        0004
  DH_P_CLASS . . . . . . . . . .        0006
  DH_C_CLASS . . . . . . . . . .        0008
  DH_V_PARENT. . . . . . . . . .        000A
  DH_V_CHILD . . . . . . . . . .        000C
  DH_MAJOR . . . . . . . . . . .        000E
  DH_MINOR . . . . . . . . . . .        000F

Segments and Groups:

                N a m e                 Size    Align   Combine Class

CODE_SEG . . . . . . . . . . . .        00A5    PARA    NONE
DATA_SEG . . . . . . . . . . . .        00A8    PARA    NONE
TS_EVENT_HEADR . . . . . . . . .        0010    PARA    NONE

Symbols:

                N a m e                 Type    Value   Attr

ATR_HP . . . . . . . . . . . . .        Number  8000
BEGIN. . . . . . . . . . . . . .        L NEAR  0000    CODE_SEG
BUTTON_PUSH. . . . . . . . . . .        L NEAR  0078    CODE_SEG
BUTTON_REPORT. . . . . . . . . .        L NEAR  0069    CODE_SEG
CL_NULL. . . . . . . . . . . . .        Number  0000
EXAM_HP_ATTR . . . . . . . . . .        Alias   ATR_HP
EXIT_PROG. . . . . . . . . . . .        L NEAR  0018    CODE_SEG
EXIT_TOUCH . . . . . . . . . . .        L NEAR  0080    CODE_SEG
F_IO_CONTROL . . . . . . . . . .        Number  0004
F_ISR. . . . . . . . . . . . . .        Number  0000
HP_ENTRY . . . . . . . . . . . .        Number  006F
INPUT_LOOP . . . . . . . . . . .        L NEAR  000E    CODE_SEG
MAKE_BREAK_BIT . . . . . . . . .        Number  0080
POS_REPORT . . . . . . . . . . .        L NEAR  005D    CODE_SEG
PROCESS_ISR. . . . . . . . . . .        L NEAR  0050    CODE_SEG
READ_CHAR_ECHO . . . . . . . . .        Number  0001
RS_SUCCESSFUL. . . . . . . . . .        Number  0000
RS_UNSUPPORTED . . . . . . . . .        Number  0002
SAVE_CS. . . . . . . . . . . . .        L WORD  0000    DATA_SEG
SAVE_DS. . . . . . . . . . . . .        L WORD  0004    DATA_SEG
SAVE_IP. . . . . . . . . . . . .        L WORD  0002    DATA_SEG
SF_CREATE_EVENT. . . . . . . . .        Number  0008
SF_EVENT_OFF . . . . . . . . . .        Number  000C
SF_EVENT_ON. . . . . . . . . . .        Number  000A
STACK. . . . . . . . . . . . . .        L WORD  0006    DATA_SEG        Length =0050
STK_TOP. . . . . . . . . . . . .        L WORD  00A6    DATA_SEG
TERMINATE_PROC . . . . . . . . .        Number  004C
TOUCH_ENABLE . . . . . . . . . .        N PROC  001D    CODE_SEG        Length =002B
TOUCH_HANDLER. . . . . . . . . .        N PROC  0048    CODE_SEG        Length =003C
TOUCH_RESTORE. . . . . . . . . .        N PROC  0084    CODE_SEG        Length =0021
T_KC_BUTTON. . . . . . . . . . .        Number  0009
T_TS . . . . . . . . . . . . . .        Number  0045
V_DOLITTLE . . . . . . . . . . .        Number  0006
V_EVENT_TOUCH. . . . . . . . . .        Number  0060
V_LTOUCH . . . . . . . . . . . .        Number  00C6

48576 Bytes free

Warning Severe
Errors  Errors
0       0
```

Hardware Interface Level

The hardware interface of the Input System is composed of a set of drivers to respond to hardware interrupts and to process physical data from the input devices into a form usable by the application interface drivers. These hardware interface level drivers are shown in Figure 4-2.

Overview

This section describes the drivers, data structures, and interrupt service routine (ISR) event processing that takes place below the application interface level. The following data flow expands on step 2 of the data flow presented previously. A detailed explanation of each step is presented after the data flow.

1. The user touches the screen. This causes a hardware interrupt which is managed by the 8259A's interrupt controller service (V_S8259). V_S8259 responds to the interrupt controller chip and transfers control to the HP-HIL driver.

2. The HP-HIL driver (V_HPHIL) services the HP-HIL controller chip, retrieving the input device data. V_HPHIL processes the input data and transfers control to the Input System dispatch service.

3. The dispatch service (V_SINPUT) transfers control to the appropriate physical device driver based on the source of the input data (in this case the physical touchscreen driver).

4. The physical touchscreen driver builds the Physical Describe Record and transfers control to the application interface driver V_LTOUCH.

V_S8259 provides a funnel point for managing HP specific hardware. The Input System hardware communicates with the hardware interface drivers via two interrupts: the 8042 service request (SVC) and the HP-HIL controller interrupt. The HP-HIL controller interrupt is chained to the HP-HIL driver (V_HPHIL); i.e., when V_S8259 receives an HP-HIL controller interrupt it generates an HP_ENTRY software interrupt to transfer control to V_HPHIL.

The HP-HIL driver services the HP-HIL controller and generates the appropriate Physical ISR Event Record(s). After processing the input data, V_HPHIL chains to V_SINPUT.

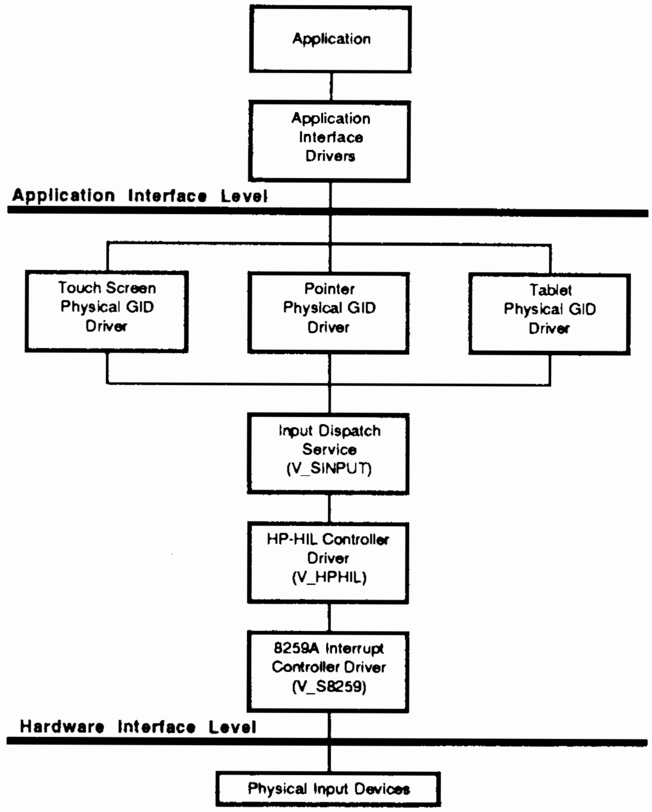


Figure 4-2. Hardware Interface Level Drivers

V_SINPUT chains to the appropriate physical device driver based on the vector index (vector address divided by six) stored in the Physical ISR Event Record (DL register). It provides an entry point into the Input System for non-HP-HIL devices. V_SINPUT also provides driver mapping functions that will be discussed later in this chapter.

Two physical drivers will be discussed later in this chapter. The first is the physical GID driver (PGID), which handles both absolute and relative data. Because PGID can handle both types of GID data, it can chain to any logical GID driver; this forms the basis for Input System device driver mapping. The second physical driver is the null device driver (V_PNULL), which serves as a handler for unsupported devices.

Device Driver Mapping

Each driver in the Input System has a vector in the HP_VECTOR_TABLE, and a driver header. Each driver header has two fields which determine the mapping of the driver. One field contains the vector of the driver's parent driver, and the other contains the vector of the driver's child driver. Refer to Chapter 2 and Appendix G for a detailed description of driver headers.

Calls are made to the vector address contained in the parent field to pass the interrupt on to the next driver in the device driver chain, moving the data from the hardware toward the application via the desired logical GID driver. Hardware commands from the application are passed down the device driver chain to the device via the vector address contained in the child vector field. By changing the value of the parent or child vector field, the sequence of drivers called to handle an interrupt or function request is changed. In general, an application may re-map a driver by changing the driver header directly. Functions are provided by the V_SINPUT service to map the physical GID drivers to the logical GID drivers.

Device Emulation

Device emulation occurs when one or more physical devices are mapped to a logical device that does not represent the original source of the data. For example, mapping a physical mouse driver to a logical touchscreen driver allows the mouse to look like a touchscreen to the application. The key requirement for a logical device driver to emulate other devices is that it accept both absolute and relative data. The logical touchscreen driver which reports absolute data must accept both absolute (touch) data and relative (mouse) data.

An example of device mapping and emulation occurring in the system is the translation of mouse input to Cursor Control keypad (CCP) input. Since standard DOS processes keyboard input only, (not mouse input), the physical GID driver which processes mouse input is mapped, in its default state, to a driver called V_PGID_CCP. This driver causes mouse input to emulate input from the CCP. For an application which processes industry standard mouse input (INT 33H) to use the HP Mouse, the mouse physical GID driver should be mapped to the installable *HP-HIL Mouse Driver* (V_LHPMOUSE), using the HP-HIL mouse driver's F33_INSTALL function. (Note that the *HP-HIL Mouse Driver* is shipped on a separate disc with all ES, QS, and RS Vectra series computers)

Data Structures

The hardware interface level uses two major data structures: the Physical Describe Record and the Physical ISR Event Record(s). These data structures help keep track of the numerous events occurring in the Input System.

Physical Describe Record

The Physical Describe Record is used by the physical GID drivers to keep track of the current state of their respective devices. Each of the physical GID drivers has a Physical Describe Record located directly after the driver header, starting with memory address DS: 0010H. Table 4-7 gives the field types and offsets of the Physical GID Device Describe Record. An explanation of the Physical Describe Record fields follows

Table 4-7. Physical GID Device Describe Record

Field Driver Header	Description Driver Header	Type	Offset 00H
D_SOURC	Input type and device address	BYTE	10H
D_HPHIL_ID	Device ID	BYTE	11H
D_DESC_MASK	Describe header byte	BYTE	12H
D_IO_MASK	Device I/O descriptor byte	BYTE	13H
D_XDESC_MASK	Extended describe header byte	BYTE	14H

Table 4-7. Physical GID Device Describe Record (Cont.)

Field Driver Header	Description Driver Header	Type	Offset 00H
D_MAX_AXIS	Maximum number of axes	BYTE	15H
D_CLASS	Device class	BYTE	16H
D_PROMPTS	Number of button/prompts	BYTE	17H
D_PARAGRAPHS	This record size in paragraphs	BYTE	18H
D_BURST_LEN	Maximum output burst length	BYTE	19H
D_WR_REG	Number of write registers	BYTE	1AH
D_RD_REG	Number of read registers	BYTE	1BH
D_TRANSITION	Button transitions	BYTE	1CH
D_STATE	Current state of the buttons	BYTE	1DH
D_RESOLUTION	Physical device resolution	WORD	1EH
D_SIZE_X	Maximum x-axis count	WORD	20H
D_SIZE_Y	Maximum y-axis count	WORD	22H
D_ABS_X	X position data for absolute devices	WORD	24H
D_ABS_Y	Y position data for absolute devices	WORD	26H
D_REL_X	X delta for relative devices	WORD	28H
D_REL_Y	Y delta for relative devices	WORD	2AH
D_ACCUM_X	Reserved	WORD	2CH
D_ACCUM_Y	Reserved	WORD	2EH
D_SIZE_Z	Maximum Z-axis count	WORD	30H
D_ABS_Z	Z position data for absolute devices	WORD	32H
D_REL_Z	Z delta for relative devices	WORD	34H
D_ACCUM_Z	Reserved	WORD	36H

Physical Device Record Definition

D_SOURCE	This field is divided into nibbles. Bits 7-4 contain the graphics input device type. This field is loaded with the low order nibble of the appropriate physical GID data type. (See Table 4-8.) Bits 3-0 are the link address of the physical device.
D_HPHIL_ID	ID byte of the physical device which last reported data. See Table 4-2 for a list of HP-HIL ID bytes.
D_DESC_MASK	Physical device describe byte. This byte contains information about the physical device characteristics. See the *HP-HIL Technical Reference Manual* for more information.
D_IO_MASK	Physical device I/O descriptor byte. This byte contains information on the number of prompts and acknowledges the device supports. See the *HP-HIL Technical Reference Manual* for more information.
D_XDESC_MASK	Physical device extended describe byte. This byte contains additional device characteristics. See HP-HIL Technical Reference Manual for more information.
D_MAX_AXIS	Maximum number of axes supported by the device. Valid range is 0-2.
D_CLASS	Device class. Bits 7-4 contain the current class. Bits 3-0 contain the default class. See Appendix G for more information on device classes.
D_PROMPTS	Number of buttons and prompts supported by the device. Bits 7-4 is the number of prompts. Bits 3-0 is the number of buttons.
D_PARAGRAPHS	Indicates size of this record in paragraphs: 0 means 3 paragraphs, 1 means 4 paragraphs.
D_BURST_LEN	Maximum number of bytes that can be output to the device using a single write command.
D_WR_REG	Number of write registers supported by the device.
D_RD_REG	Number of read registers supported by the device.
D_TRANSITION	Transitions reported per button; i.e. a set bit indicates that the corresponding button was either pushed or released. Bit 7 corresponds to button 7, etc.
D_STATE	Current state of the buttons. 0 is down, 1 is up. Bit 7 corresponds to button 7, etc. If D_STATE is XOR'ed with D_TRANSITION the result is the previous button state.
D_RESOLUTION	This is the resolution of the physical device. The resolution is in counts per meter for devices that report 8 bits of data. For devices that report 16 bits of data, the resolution is in counts per centimeter.
D_SIZE_X	Maximum count (in units of resolution) for the x-axis.
D_SIZE_Y	Maximum count (in units of resolution) for the y-axis.

D_ABS_X — X position data for devices which report absolute coordinates (absolute devices).

D_ABS_Y — Y position data for devices which report absolute coordinates.

D_REL_X — Latest change in x position for devices which return coordinates relative to the previous position (relative devices).

D_REL_Y — Latest change in y position for devices which return coordinates relative to the previous position.

D_SIZE_Z — Maximum count (in units of resolution) for the z-axis .

D_ABS_Z — Z position data for devices which report absolute coordinates.

D_REL_Z — Latest change in z position for devices which return coordinates relative to the previous position (relative devices).

Physical ISR Event Records

A Physical ISR Event Record is not a data structure in the truest sense, but is a set of register definitions for inter-driver communication of input events. The following define the Physical ISR Event Records.

GID Button ISR Event Record

```
    AH = F_ISR (00H)
    DL = Physical device driver's vector address / 6
    BX = Button information.

         Bit      Value   Definition

         0FH-08H          Reserved
         07H        1     Button up
                    0     Button down
         06H-00H          Button number (0-7)

    DH = Data Type
  ES:0 = Pointer to physical device driver
         header and Physical Describe Record.
```

GID Motion ISR Event Record

```
    AH = F_ISR (00H)
    DL = Physical device driver's vector address / 6
    BX = X axis motion in raw data form.
    CX = Y axis motion in raw data form.
    SI = Z axis motion in raw data form.
    DH = Data Type
  ES:0 = Pointer to physical device driver
         header and Physical Describe Record.
```

The button number in the Button Transition Information field (BX) denotes which button on the device is reporting data. Of special interest is button seven (proximity indicator), which is currently used by absolute devices to indicate that the device measurement field is active; ie., someone is touching the touchscreen, or the stylus is in contact with the tablet surface.

The Data Type field (DH) contains a code representing the current type of physical GID data stored in the event record. For button events, this value will be T_KC_BUTTON. For a complete list of physical GID event data types see Table 4-8.

Table 4-8. Physical GID Event Data Types

Type	Value	Definition
T_KC_BUTTON	09H	Button data.
T_REL08	40H	Signed 8 bit relative data
T_REL16	41H	Signed 16 bit relative data
T_ABS08	42H	Unsigned 8 bit absolute data
T_ABS16	43H	Unsigned 16 bit absolute data

Hardware Interface Level Drivers

This section describes the hardware interface level drivers in detail.

V_S8259 Driver (BP = 001EH)

The V_S8259 driver services the HP interrupt. Three interrupt sources will genterate this interrupt: the 8042 SVC (Service port) service request, the HP-HIL controller, and the 8042 SCAN interrupt.

When an HP interrupt occurs, the V_S8259 driver will determine the source of the interrupt and perform an F_ISR call to one of the three drivers:

- the V_8042 driver for an 8042 SVC interrupt,
- the V_HPHIL driver for an HP-HIL controller interrupt,
- the V_SCANDOOR driver for a SCAN interrupt.

In addition to initiating response to the hardware interrupts, the 8259A driver has other functions which initialize the interrupt vectors and program the proper parameters into the 8259A interrupt controllers.

V_S8259 Driver Function Definitions

A summary of the V_S8259 function codes is provided in Table 4-9.

Table 4-9. V_S8259 Function Code Summary

Function Equate	Definition	Vector Address	Func. Value
V_S8259	8259 interrupt controller support	001EH	
F_SYSTEM	System functions	001EH	02
SF_INIT	Initialize HP-HIL IRQ	001EH	02/00
SF_START	Enable HP-HIL interrupts	001EH	02/02
SF_VERSION _DESC	Report HP version number	001EH	02/06
SF_GET_IRQ	Get HP IRQ number	001EH	04/14

F_ISR (AH = 00H)

Because this driver directly services hardware interrupts from an 8259A interrupt controller, this function is not applicable. If called, this function will return a Return Status Code of RS_UNSUPPORTED.

SF_INIT (AX = 0200H)

This subfunction sets the interrupt vectors for the HP-HIL IRQ (default IRQ 12). This subfunction leaves interrupts disabled. They must be enabled with the SF_START subfunction.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BP = V_S8259 (001EH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_START (AX = 0202H)

This subfunction enables the HP-HIL interrupts.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_START (02H)
          BP = V_S8259 (001EH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960, and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_S8259 (001EH)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

SF_GET_IRQ (AX = 0414H)

This function gets the current IRQ number associated with the SCAN/STATE/HIL/SVC interrupts.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_GET_IRQ (14H)
          BP = V_S8259 (001EH)

On Exit: AH = RS_SUCCESSFUL (00H)
         BL = Current IRQ

Registers Altered: AX, BX, BP, DS
```

V_HPHIL Driver (BP = 0114H)

The HP-HIL driver retrieves input data from the HP-HIL controller and builds an ISR Event Record to pass to V_SINPUT.

A summary of the V_HPHIL driver function codes is provided in Table 4-10.

Table 4-10. V_HPHIL Driver Function Code Summary

Function Equate	Definition	Vector Address	Func. Value
V_HPHIL	Set up HP-HIL to INPUT driver linkage	0114H	
F_ISR	Logical Interrupt	0114H	00
F_SYSTEM	System Functions	0114H	02
SF_INIT	Initializes the driver data area.	0114H	02/00
SF_REPORT _STATE	Reports state of device	0114H	02/04
SF_VERSION _DESC	Reports driver version number.	0114H	02/06
SF_OPEN	Put driver in open state.	0114H	02/0E
SF_CLOSE	Put driver in closed state.	0114H	02/10
F_IO_CONTROL	I/O control to driver	0114H	04
SF_CRV_CRV _MAJ_MIN	Reserved	0114H	04/04
SF_CRV _RECONFIGURE	Forces HP-HIL to reconfigure all devices.	0114H	04/06
SF_CRV_WR _PROMPTS	Writes a prompt to a device	0114H	04/08
SF_CRV_WR _ACK	Writes an acknowledge to a device	0114H	04/0A
SF_CRV _REPEAT	Sets either 30Hz or 60Hz repeat rate	0114H	04/0C
SF_CRV _DISABLE _REPEAT	Cancels keyboard repeat rate	0114H	04/0E

Table 4-10. V_HPHIL Driver Function Code Summary (Cont.)

Function Equate	Definition	Vector Address	Func. Value
SF_CRV _SELF_TEST	Issues self-test command to physical device.	0114H	04/10
SF_CRV _REPORT _STATUS	Gets status from any HP-HIL device that needs to report	0114H	04/12
SF_CRV _REPORT_NAME	Returns the ASCII name for a device	0114H	04/14
SF_GET _DEVTBL	Gets physical device table address	0114H	04/20
SF_SET _DEVTBL	Sets physical device table address	0114H	04/22
SF_DEF _DEVTBL	Sets default physical device table	0114H	04/24
F_PUT_BYTE	Writes one byte to specified HP-HIL device.	0114H	06
F_GET_BYTE	Reads one byte from specified HP-HIL device.	0114H	08
F_PUT_BUFFER	Writes a string of bytes to HP-HIL device.	0114H	0A

V_HPHIL Driver Function Definitions

F_ISR (AH = 00H)

This function is called by the V_S8259 driver to initiate processing of an interrupt from the HP-HIL controller. This function reads input device data from the HP-HIL controller, generates one or more ISR Event Records, and chains to V_SINPUT. THIS FUNCTION SHOULD ONLY BE CALLED BY THE V_S8259 DRIVER.

```
On Entry: AH = F_ISR (00H)
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_INIT (AX = 0200H)

This subfunction initializes the driver and HP-HIL controller. Refer to Chapter 8 for a complete discussion of the protocol utilized in data space allocation ("last used DS" passed in register BX).

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BX = "Last used DS" in HP Data Area
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code
         BX = New "last used DS" in
              HP Data Area

Registers Altered: AX, BX, BP, DS
```

SF_REPORT_STATE (AX = 0204H)

This subfunction returns the current status of V_HPHIL.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_REPORT_STATE (04H)
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code
         BX = Status word
```

Bit	Value	Definition
0FH		Reserved
0EH	1	HP-HIL is OFF
	0	HP-HIL is ON
0DH		Reserved
0CH	1	Timeout has occurred
0BH	1	Output request has completed
0AH		Reserved
09H	1	Error during output request
08H	1	HP-HIL link has been reconfigured
07H		Reserved
06H	1	HP-HIL driver is open
	0	HP-HIL driver is closed
05H-04H		Reserved
03H	1	General failure
02H	1	No devices attached.
01H		Reserved
00H	1	Link configuration in progress

Registers Altered: AX, BX, BP, DS

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number
```

Registers Altered: AX, BX, CX, DI, ES, BP, DS

SF_OPEN (AX = 020EH)

This subfunction puts the HP-HIL driver in the open state. When the driver has been placed in the open state, output to the HP-HIL devices is allowed.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_OPEN (0EH)
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code
```

Registers Altered: AX, BP, DS

SF_CLOSE (AX = 0210H)

This subfunction puts the HP-HIL driver in the closed state. When the driver has been placed in the closed state, output to the HP-HIL devices is not allowed.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_CLOSE (10H)
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_CRV_RECONFIGURE (AX = 0406H)

This subfunction instructs the HP-HIL controller to reconfigure the link.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CRV_RECONFIGURE (06H)
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_CRV_WR_PROMPTS (AX = 0408H)

This subfunction issues a prompt command to a device on the HP-HIL link. The prompt command is either specific (prompt number 1-7) or generic (a prompt number other than 1-7).

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CRV_WR_PROMPTS (08H)
          BX = Device address indicator

            Bit     Value Definition

            0FH-0EH       Reserved
            0DH       1   Valid address is present in DH
                      0   Reserved for future enhancement,
                          currently returns RS_FAIL
            0CH       1   Valid register is present in DL
            0BH-00H       Reserved

          DH = HP-HIL device address
          DL = Prompt number
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF__CRV__WR __ACK (AX = 040AH)

This subfunction issues an acknowledge command to a device on the HP-HIL link. The acknowledge command is either specific (acknowledge number 1-7) or generic (an acknowledge number other than 1-7).

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CRV_WR_ACK (0AH)
          BX = Device address indicator

            Bit      Value Definition

            0FH-0EH        Reserved
            0DH        1   Valid address is present in DH
                       0   Reserved for future enhancement,
                           currently returns RS_FAIL
            0CH        1   Valid register is present in DL
            0BH-00H        Reserved

          DH = HP-HIL device address (major address)
          DL = Acknowledge number
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF__CRV __REPEAT (AX = 040CH)

This subfunction sets the key repeat rate of a specific HP-HIL device. A repeat rate of 30 or 60 times a second may be specified. This subfunction will operate only if the HP-HIL driver is in the open state.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CRV_REPEAT (0CH)
          BX = Device address indicator

             Bit      Value  Definition

             0FH-0EH         Reserved.
             0DH        1    Valid address is present in DH.
                        0    Reserved for future enhancement,
                             currently returns RS_FAIL.
             0CH        1    Valid register is present in DL.
             0BH-00H         Reserved.

          CL = 0 for a repeat rate of 30 Hz, 1 for 60 Hz
          DH = HP-HIL device address (major address)
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_CRV_DISABLE_REPEAT (AX = 040EH)

This subfunction disables the key repeat of a specified HP-HIL device. This subfunction will operate only if the HP-HIL driver is in the open state.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CRV_DISABLE_REPEAT (0EH)
          BX = Device address indicator

             Bit       Value   Definition

             0FH-0EH           Reserved
             0DH         1      Valid address is present in DH.
                         0     Reserved for future enhancement,
                               currently returns RS_FAIL.
             0CH         1     Valid register is present in DL.
             0BH-00H           Reserved

          DH = HP-HIL device address (major address)
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_CRV_SELF_TEST (AX = 0410H)

This subfunction initiates device self-test on the specified HP-HIL device. The HP-HIL device will respond with a one byte status code indicating the result of the test. This subfunction should not be called with an HP-HIL device address of zero (all devices), as the test could then take up to 1.5 seconds to execute. Also, if one of the devices fails, there would be no way to determine which device reported a failure.

On exit, the buffer has the return status of the self-test done on the physical device.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CRV_SELF_TEST (10H)
          BX = Device address indicator

             Bit      Value   Definition

             0FH-0EH          Reserved
             0DH        1     Valid address is present in DH
                        0     Reserved for future enhancement,
                              currently returns RS_FAIL
             0CH        1     Valid register is present in DL
             0BH-00H          Reserved

          DH = HP-HIL device address (major address)
          BP = V_HPHIL (0114H)
       ES:SI = Pointer to a buffer area
```

```
On Exit:  AH = Return Status Code
       ES:SI = Pointer to buffer area
          CX = Number of bytes in buffer

Registers Altered: AX, CX, BP, DS
```

SF_CRV_REPORT_STATUS (AX = 0412H)

This subfunction issues a send status command to a specified HP-HIL device. The returned status information ranges from 1 to 15 bytes in length. A pointer to a 15 byte buffer must be passed to the subfunction. This subfunction will operate only if the HP-HIL driver is in the open state.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CRV_REPORT_STATUS (12H)
          BX = Device address indicator

             Bit      Value   Definition

             0FH-0EH          Reserved
             0DH        1     Valid address is present in DH.
                        0     Reserved for future enhancement,
                              currently returns RS_FAIL.
             0CH        1     Valid register is present in DL.
             0BH-00H          Reserved

          DH = HP-HIL device address (major address)
          BP = V_HPHIL (0114H)
       ES:SI = Pointer to a buffer area

On Exit:  AH = Return Status Code
       ES:SI = Pointer to buffer area
          CX = Number of bytes in buffer

Registers Altered: AX, CX, BP, DS
```

SF_CRV_REPORT_NAME (AX = 0414H)

This subfunction issues a report name command to a specified HP-HIL device. The returned name information ranges from 1 to 15 bytes in length. A pointer to a 15 byte buffer must be passed to the subfunction. This subfunction will operate only if the HP-HIL driver is in the open state.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CRV_REPORT_NAME (14H)
          BX = Device address indicator

             Bit      Value   Definition

             0FH-0EH          Reserved
             0DH        1     Valid address is present in DH.
                        0     Reserved for future enhancement,
                              currently returns RS_FAIL.
             0CH        1     Valid register is present in DL.
             0BH-00H          Reserved

          DH = HP-HIL device address (major address)
          BP = V_HPHIL (0114H)
       ES:SI = Pointer to a buffer area

On Exit:  AH = Return Status Code
       ES:SI = Pointer to buffer area
          CX = Number of bytes in buffer

Registers Altered: AX, CX, BP, DS
```

F_PUT_BYTE (AH = 06H)

This function outputs a byte of data to a specific HP-HIL device register. This function will operate only if the HP-HIL driver is in the open state.

```
On Entry: AH = F_PUT_BYTE (06H)
          AL = Byte to output
          BX = Device address indicator

             Bit      Value   Definition

             0FH-0EH          Reserved
             0DH        1     Valid address is present in DH.
                        0     Reserved for future enhancement,
                              currently returns RS_FAIL.
             0CH        1     Valid register is present in DL.
             0BH-00H          Reserved

          DH = HP-HIL device address
          DL = HP-HIL device register (0-127)
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code
```

Registers Altered: AX, BP, DS

F_GET_BYTE (AH = 08H)

This function returns the contents of a specific HP-HIL device register. This function will operate only if the HP-HIL driver is in the open state.

```
On Entry: AH = F_GET_BYTE (08H)
          BX = Device address indicator

             Bit      Value  Definition

             0FH-0EH         Reserved
             0DH        1    Valid address is present in DH.
                        0    Reserved for future enhancement,
                             currently returns RS_FAIL.
             0CH        1    Valid register is present in DL.
             0BH-00H         Reserved

          DH = HP-HIL device address
          DL = HP-HIL device register (0-127)
          BP = V_HPHIL (0114H)

On Exit: AH = Return Status Code
         AL = Contents of specified register
```

Registers Altered: AX, BP, DS

F_PUT_BUFFER (AH = 0AH)

This function outputs a buffer to a specific HP-HIL device register. The HP-HIL controller and devices are capable of data transfer at rates up to 6500 bytes per second. If the number of bytes in the buffer is greater than the number the HP-HIL device can handle, this function will transfer as many bytes as possible to the device, and adjust the value in CX to reflect the number of bytes left in the buffer (not sent to the device).

```
On Entry: AH = F_PUT_BUFFER (0AH)
          BX = Device address indicator

             Bit      Value  Definition

             0FH-0EH         Reserved
             0DH        1    Valid address is present in DH.
                        0    Reserved for future enhancement,
                             currently returns RS_FAIL.
             0CH        1    Valid register is present in DL.
             0BH-00H         Reserved

          CX = Number of bytes in buffer
          DH = HP-HIL device address
          DL = HP-HIL device register (0-127)
          BP = V_HPHIL (0114H)
```

```
        ES:SI = Pointer to buffer containing data to output

On Exit: AH = Return Status Code
         CX = 0 means all the data in buffer is transferred,
              otherwise the number of bytes left in buffer.

Registers Altered: AX, CX, BP, DS
```

SF_GET_DEVTBL (AX = 0420H)

This function returns the address and size of the physical device table (listed in Table 4-11).

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_GET_DEVTBL (20H)
          BP = V_HPHIL (0114H)

On Exit:  AH = RS_SUCCESSFUL (00H)
          DS:SI = Address of current physical device table
          CX = Number of table entries

Registers Altered: AX, CX, SI, BP, DS
```

Table 4-11. Physical Device Table

Field	Type	Offset	Size	Description
P_ID_LOWER	Byte	00H	1	HPHIL ID lower bound
P_ID_UPPER	Byte	01H	1	HPHIL ID up-per bound
P_OFFSET	Word	02H	1	Offset of driver entry point
P_CS	Word	04H	1	Segment of driver entry point
P_HEADER	Byte	06H	16	Header for physical driver
P_CLASS	Byte	16H	1	Device driver class: Bits 7-4 current class Bits 3-0 default class

Table 4-11. Physical Device Table (Cont.)

Field	Type	Offset	Size	Description
P_TYPE	Byte	17H	1	ISR event record type
P_EXTRA_DS	Word	18H	1	Pointer to Extra DS maintained by the device driver

Both the SF_GET_DEVTBL and SF_SET_DEVTBL are intended to be used by installable HP-HIL device drivers that need to provide their own physical describe record. For the HP Vectra series of computers, the installable HP-HIL device driver can request the address and function, copy the table to local RAM, add any special entries it needs to the table, and then set the new table's address by issuing the SF_SET_DEVTBL function. The advantage of this is that once the HP-HIL device driver is installed, and its new entries added into the table, it will always be recognized by the system even during a loop reconfiguration.

The P_EXTRA_DS is for the device drivers use. It should hold the segment address of any additional data area that the device may require. This field (P_EXTRA_DS) will not be altered by the system when reconfiguring the HP-HIL loop.

SF_SET_DEVTBL (AX = 0422H)

This function sets the new address and size of the physical device table.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_SET_DEVTBL (20H)
          BP = V_HPHIL (0114H)
          ES:DI = Address of a physical device table
          CX = Number of entries in table

On Exit:  AH = RS_SUCCESSFUL (00H)

Registers Altered: AX, CX, SI, BP, DS
```

SF_SET_DEVTBL (AX = 0424H)

This function resets the physical device table to default power-on values.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_DEF_DEVTBL (24H)
          BP = V_HPHIL (0114H)

On Exit:  AH = RS_SUCCESSFUL (00H)

Registers Altered: AX, BP, DS
```

V_SINPUT (BP = 002AH)

The V_SINPUT driver dispatches ISR events generated by the HP-HIL controller to the appropriate physical driver, thus providing an entry point into the Input System for non-HP-HIL devices (i.e., RS-232 mice, tablets, etc.). It also provides a number of functions which support device mapping.

A summary of the V_SINPUT driver function codes is provided in Table 4-12.

Table 4-12. V_SINPUT Driver Function Code Summary

Function Equate	Definition	Vector Address	Func. Value
V_SINPUT	Inquire Commands	002AH	
F_ISR	Pass ISR event record to physical driver	002AH	00
F_SYSTEM	System Functions	002AH	02/
SF_INIT	Initialize driver	002AH	02/00
F_IO_CONTROL	Entry point to IO control functions	002AH	04
SF_DEF_LINKS	Set header link fields to system defaults	002AH	04/00
SF_GET_LINKS	Return device header link field entries	002AH	04/02
SF_SET_LINKS	Set device header link field entries	002AH	04/04
F_INQUIRE	Return describe record for an HP-HIL device.	002AH	06
F_INQUIRE_ALL	Return device IDs for all HP-HIL devices present	002AH	08
F_INQUIRE_FIRST	Return vector address of first HP-HIL device driver.	002AH	0A
F_REPORT_ENTRY	Report entry point of PGID	002AH	0C

V_SINPUT Driver Function Definitions

F_ISR (AH = 00H)

This function passes an ISR Event Record to the appropriate physical device driver based on the value in DL. Non-HP-HIL devices which call V_SINPUT must provide the physical device driver that will handle the ISR event record, and must place its vector index (vector address divided by six) in DL. (See Chapter 8, V_SYSTEM functions, to obtain a valid vector address).

```
On Entry: AH = F_ISR (00H)
          BP = V_SINPUT
          (See tables 4-6 and 4-7 for other register values)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```


SF_INIT (AX = 0200H)

This subfunction initializes the driver.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BP = V_SINPUT (002AH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_DEF _LINKS (AX = 0400H)

This subfunction sets the parent vectors in the HP-HIL physical device driver headers to the system defaults shown in Table 4-13. The child vector entries are set to the null device driver (V_PNULL) by default (see Appendix F).

Table 4-13. Default Physical Device Driver Parents

Device	Parent
Mouse	V_PGID_CCP
Tablet	V_LTABLET
Touchscreen	V_LTOUCH
Barcode Reader	V_PNULL
Rotary Knob	V_PGID_CCP

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_DEF_LINKS (00H)
          BP = V_SINPUT (002AH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_GET_LINKS (AX = 0402H)

This subfunction returns the current parent and child vectors in the HP-HIL physical device driver headers. The address of a seven word (14 byte) table is passed to the subfunction. When the subfunction returns, the buffer will contain the current vectors. See Table 4-14 for the mapping buffer format.

Table 4-14. Mapping Buffer Format

Word	Parent Vector	Child Vector	HP-HIL Device
0	High byte	Low byte	Device # 1
1	" "	" "	" " 2
2	" "	" "	" " 3
3	" "	" "	" " 4
4	" "	" "	" " 5
5	" "	" "	" " 6
6	" "	" "	" " 7

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_GET_LINKS (02H)
          BP = V_SINPUT (002AH)
       ES:SI = Pointer to table

On Exit: AH = Return Status Code
      ES:SI = Pointer to table

Registers Altered: AX, BP, DS
```

SF_SET_LINKS (AX = 0404H)

This subfunction sets the parent and child vectors in the HP-HIL physical device driver headers. The address of a seven word (14 byte) table is passed to the subfunction. The table contains the new parent and child vectors for the drivers. The format of the buffer is shown in Table 4-14.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_SET_LINKS (04H)
          BP = V_SINPUT (002AH)
       ES:SI = Pointer to table

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

The following example is how to use the SF_SET_LINKS function. It is presumed that a call to F_INQUIRE_ALL has been made, and that the device is a tablet. The tablet is going to be mapped to the installable HP-HIL Mouse driver (V_LHPMOUSE). The BX register already has the offset into the buffer of tablet mappings.

```
BUFFER  DW  7 DUP (?)
     MOV CX, BUFFER[BX]      ; get the current mapping of the tablet
     MOV CH, V_LHPMOUSE / 6  ; change tablet to HP Mouse
     MOV BUFFER[BX], CX      ; save the new mapping
     MOV AH, F_IO_CONTROL    ; load function code
     MOV AL, SF_SET_LINKS    ; load subfunc. code
     MOV BP, V_SINPUT        ; load vector address
     LEA SI, BUFFER          ; get the offset of the buffer
     PUSH DS
     POP ES                  ; ES = DS
     PUSH DS                 ; save current DS
     CALL SYSCALL            ; call extended BIOS driver
     POP DS
```

F_INQUIRE (AH = 06H)

This function returns a pointer to the Physical Describe Record of the specified HP-HIL physical device driver.

WARNING

The Physical Describe Record should not be modified in any way.

```
On Entry: AH = F_INQUIRE (06H)
          AL = HP-HIL Device Number (1-7)
          BP = V_SINPUT (002AH)

On Exit:  AH = Return Status Code
       ES:SI = Pointer to Physical Describe Record

Registers Altered: AX, BP, SI, DS, ES
```

F_INQUIRE_ALL (AH = 08H)

This subfunction is used to determine which HP-HIL devices are present on the loop. The address of a seven-word table is passed to the subfunction. When the subfunction returns, the table will contain the current status of all HP-HIL devices. The format of the Device Inquire buffer is shown in Table 4-15.

Table 4-15. Device Inquire Buffer Format

Word	HP-HIL Device ID	Device Status*	HP-HIL Device
0	High byte	Low byte	Device # 1
1	" "	" "	" " 2
2	" "	" "	" " 3
3	" "	" "	" " 4
4	" "	" "	" " 5
5	" "	" "	" " 6
6	" "	" "	" " 7

* Bit 0 = 1 if device present, 0 if no device at this address.
Bits 2 - 7 are reserved.

```
On Entry: AH = F_INQUIRE_ALL (08H)
          BP = V_SINPUT (002AH)
       ES:SI = Pointer to table

On Exit: AH = Return Status Code
      ES:SI = Pointer to table

Registers Altered: AX, BP, DS
```

The following example shows how to use the F__INQUIRE__ALL function.

```
BUFFER DW 7 DUP (?)
     MOV  AH, F_INQUIRE_ALL ; load function code
     MOV BP, V_SINPUT       ; load vector address
     LEA  SI, BUFFER        ; get offset of buffer
     PUSH  DS
     POP  ES                ; ES = DS
     PUSH  DS               ; save current DS
     CALL SYSCALL           ; call EX-BIOS driver
     POP DS                 ; restore DS
```

F__INQUIRE__FIRST (AH = 0AH)

This function returns the vector address of the first HP-HIL physical device driver (HP-HIL address 1). This address allows the vector address of all HP-HIL physical device drivers to be easily calculated since the vectors are contiguous in the HP__VECTOR__TABLE (see Table 4-16).

```
On Entry: AH = F_INQUIRE_FIRST (0AH)
          BP = V_SINPUT (002AH)

On Exit: AH = Return Status Code
         BX = Vector address of first HP-HIL physical device driver

Registers Altered: AX, BX, BP, DS
```

F_REPORT _ENTRY (AH = 0CH)

This function is used to get the CS:IP of the physical GID driver.

```
On Entry: AH = F_REPORT_ENTRY (0CH)
          BP = V_SINPUT (002AH)

On Exit: AH = Return Status Code
         BX = offset of physical GID driver
         ES = segment of physical GID driver

Registers Altered: AX, BX, BP, DS, ES
```

Physical GID Driver

The physical GID driver is responsible for updating the Physical Describe Record. Two types of graphics input devices are defined in the input system, absolute (touchscreen and tablet), and relative (mouse). An instance of this driver (same code module, different data area) is installed for each graphic input device present.

A summary of the PGID function codes is provided in Table 4-16.

Table 4-16. Physical GID Driver Function Code Summary

Func. Value	Function Equate	Definition
xxxH		HP-HIL driver vector 1 through HP-HIL driver vector 7. Physical HP-HIL driver vectors (these vectors do not have fixed HP_VECTOR_TABLE addresses)
00	F_ISR	Logical Interrupt
02	F_SYSTEM	System functions
02/00	SF_INIT	Initialize driver
02/02	SF_START	Start driver
02/04	SF_REPORT _STATE	Unsupported
02/06	SF_VERSION _DESC	Report HP version number

Physical GID Driver Function Definitions

F_ISR (AH = 00H)

This function processes ISR Event Records, updates the fields in its Physical Describe Record, and then calls its parent driver. HP-HIL devices report upward relative motion with a positive sign and downward relative motion with a negative sign. The industry standard representation is the opposite of this.

```
On Entry: AH = F_ISR (00H)
          DH = Data Type
          DL = Physical device driver's vector address / 6
          BP = HP-HIL device n vector address

          For Button Event:

          BX = Button information.

             Bit       Value Definition

             0FH-08H    -    Reserved
             07H        1    Button up
                        0    Button down
             06H-00H    -    Button number (0-7)

          For Motion Event:

          BX = X axis motion in raw data form.
          CX = Y axis motion in raw data form.

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_INIT (AX = 0200H)

This subfunction is called to initialize the driver.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BP = HP-HIL device n vector address

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_START (AX = 0202H)

This subfunction starts the driver.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_START (02H)
          BP = HP-HIL device n vector address

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960, and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = HP-HIL device n vector address

On Exit: AH = Return status code
         BX = Release date code
         CX = Number of byte in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

V_PNULL Driver (BP = 000CH)

The null device driver is the default event driver routine. It is used when the physical device is not recognized or the user event handler is not installed. It.sets the AH register to RS_SUCCESSFUL and does an IRET.

Hardware Interface Level Services

Service drivers are provided as useful subroutines available to any driver. Currently the hardware interface level has only one service, the tracking sprite, V_STRACK.

V_STRACK Driver (BP = 005AH)

V_STRACK is called by the logical GID drivers to move the graphics cursor (sprite) on the display screen. V_STRACK provides functions that allow the parameters of the sprite to be defined, and move the sprite around the display.

A summary of the V_STRACK function codes is provided in Table 4-17.

Table 4-17. V_STRACK Driver Function Code Summary

Function Equate	Definition	Vector Address	Function Value
V_STRACK	Sprite control	005AH	
F_SYSTEM	System functions	005AH	02
SF_INIT	Initialize driver	005AH	02/00
SF_START	Start driver	005AH	02/02
F_TRACK_INIT	Sets tracking to default state	005AH	04
F_TRACK_ON	Enables tracking	005AH	06
F_TRACK_OFF	Disables tracking	005AH	08
F_DEF_MASKS	Define sprite masks	005AH	0A
F_SET_LIMITS_X	Set max/min horizontal values	005AH	0C
F_SET_LIMITS_Y	Set max/min vertical values	005AH	0E
F_PUT_SPRITE	Display sprite	005AH	10
F_REMOVE _SPRITE	Remove sprite from display	005AH	12

V_STRACK Driver Function Definitions

F_ISR (AH = 00H)

This function is called to move the sprite to a new location. The display under the sprite is restored, and the sprite is redisplayed in its new location. The hot spot of the sprite is placed at the coordinates passed in BX and CX.

```
On Entry: AH = F_ISR (00H)
          BX = X coordinate of sprite
          CX = Y coordinate of sprite
          DL = Source vector index
          BP = V_STRACK (005AH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_INIT (AX = 0200H)

This subfunction is called to initialize the driver. Refer to Chapter 8 for a complete discussion of the protocol utilized in data space allocation ("last used DS" passed in register BX).

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BX = "Last used DS" in HP Data Area
          BP = V_STRACK (005AH)

On Exit: AH = Return Status Code
         BX = New "last used DS" in HP Data Area

Registers Altered: AX, BX, BP, DS
```

SF_START (AX = 0202H)

This subfunction is called to start the tracking driver.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_START (02H)
          BP = V_STRACK (005AH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

F_TRACK_INIT (AH = 04H)

This function sets the tracking driver to its default state. It determines the current video mode and initializes the tracking parameters.

```
On Entry: AH = F_TRACK_INIT (04H)
          BP = V_STRACK (005AH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

F_TRACK_ON (AH = 06H)

This function enables tracking. The sprite is displayed on the screen.

```
On Entry: AH = F_TRACK_ON (06H)
          BP = V_STRACK (005AH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

F_TRACK_OFF (AH = 08H)

This function disables tracking. The sprite is removed from the screen.

```
On Entry: AH = F_TRACK_OFF (08H)
          BP = V_STRACK (005AH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

F_DEF_MASKS (AH = 0AH)

This function is called to define the sprite and screen masks used by the driver. If tracking is enabled, the sprite is erased and the new sprite is displayed in its place. The size of the sprite (its width in bytes multiplied by its height) is limited to a total of 144 bytes. The width of the save area is one byte greater than the width of the sprite.

```
On Entry: AH = F_DEF_MASKS (0AH)
          BH = Width of the save area (in bytes)
          BL = Hot Spot X coordinate
          CH = Height of sprite (in scan lines)
          CL = Hot Spot Y coordinate
          BP = V_STRACK (005AH)
          ES:SI = Pointer to sprite mask

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

The following example shows how to use the F_DEF_MASKS function provided by the tracking driver.

```
SPRITE  DW 0F9FFH ; 1111100111111111 * marks
        DW 0F0FFH ; 11110*0011111111 the Hot
        DW 0E07FH ; 1110000001111111 Spot
        DW 0E07FH ; 1110000001111111
        DW 0C03FH ; 1100000000111111
        DW 0C03FH ; 1100000000111111
        DW 0801FH ; 1000000000011111
        DW 0801FH ; 1000000000011111
        DW 0000FH ; 0000000000001111
        DW 0000FH ; 0000000000001111
        DW 0F0FFH ; 1111000011111111
        DW 0F0FFH ; 1111000011111111
        DW 0F0FFH ; 1111000011111111
        DW 0F0FFH ; 1111000011111111
        DW 0F0FFH ; 1111000011111111
        DW 0F0FFH ; 1111000011111111
```

```
                    ; Define the XOR mask
                    ;
        DW 00000H ; 0000000000000000 * marks
        DW 00600H ; 00000*1000000000 the Hot
        DW 00F00H ; 0000111100000000 Spot
        DW 00F00H ; 0000111100000000
        DW 01F80H ; 0001111110000000
        DW 01F80H ; 0001111110000000
        DW 03FC0H ; 0011111111000000
        DW 03FC0H ; 0011111111000000
        DW 07FE0H ; 0111111111100000
        DW 00600H ; 0000011000000000
        DW 00600H ; 0000011000000000
        DW 00600H ; 0000011000000000
        DW 00600H ; 0000011000000000
        DW 00600H ; 0000011000000000
        DW 00600H ; 0000011000000000
        DW 00000H ; 0000000000000000

    MOV AH, F_DEF_MASKS ; load function code
    LEA SI, SPRITE      ; get the offset of the sprite
    PUSH DS
    POP ES              ; ES = DS of sprite
    MOV CH, 10H         ; height of sprite
    MOV BH, 3           ; number of bytes wide of the save area
    MOV BL, 5           ; hot spot x
    MOV CL, 1           ; hot spot y
    MOV BP, V_STRACK    ; load vector address
    PUSH DS             ; save current DS
    CALL SYSCALL        ; call EX-BIOS DRIVER
    POP DS              ; restore DS
```

F_SET_LIMITS_X (AH = 0CH)

This function sets the minimum and maximum horizontal position of the sprite on the screen. The default minimum and maximum values are the same as the current screen mode.

```
On Entry: AH = F_SET_LIMITS_X (0CH)
          CX = Minimum X coordinate
          DX = Maximum X coordinate
          BP = V_STRACK (005AH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

F_SET_LIMITS_Y (AH = 0EH)

This function sets the minimum and maximum vertical position of the sprite on the screen. The default minimum and maximum values are the same as the current screen mode.

```
On Entry: AH = F_SET_LIMITS_Y (0EH)
          CX = Minimum Y coordinate
          DX = Maximum Y coordinate
          BP = V_STRACK (005AH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

F_PUT_SPRITE (AH = 10H)

This function is called to put the sprite on the display.

```
On Entry: AH = F_PUT_SPRITE (10H)
          BX = X coordinate of sprite
          CX = Y coordinate of sprite
          BP = V_STRACK (005AH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

F_REMOVE_SPRITE (AH = 12H)

This function removes the sprite from the display.

```
On Entry: AH = F_REMOVE_SPRITE (12H)
          BP = V_STRACK (005AH)

On Exit: AH = Return Status Code.

Registers Altered: AX, BP, DS
```

V__SCANDOOR Driver (BP = 016EH)

The V__SCANDOOR driver allows scancodes from the keyboard to be routed to the EX-BIOS before being sent to the 8042 data port (60H). A summary of the SCANDOOR driver function codes is provided in Table 4-18.

Table 4-18. V__SCANDOOR Driver Function Code Summary

Function Equate	Definition	Vector Address	Function Value
V__SCANDOOR	SCANDOOR Driver	016EH	
F__ISR	Process SCANDOOR interrupt	016EH	00
F__SYSTEM	System function	016EH	02
SF__INIT	Initialize driver	016EH	02/00
SF__START	Driver start-up	016EH	02/02
SF__VERSION__DESC	Reports HP version number	016EH	02/06
F__STATE__IOCTL	STATE functions	016EH	08
SF__GET__STATE	Get a STATE byte	016EH	08/00

V__SCANDOOR Driver Function Definitions

F__ISR (AH = 00H)

This function is called by the V__S8259 driver to initiate processing of a hardware interrupt from the 8042.

CAUTION

This function should not be called directly by an application program.

```
On Entry: AH = F_ISR (00H)
          BP = V_SCANDOOR (016EH)

On Exit:  AH = Return status

Registers Altered: AX, BP, DS
```

SF__INIT (AX = 0200H)

This subfunction initializes the driver. The driver will allocate and initialize local and global memory that belongs to it and prepare itself for start-up.

CAUTION

This function should not be called directly by an application program.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BX = Last used DS
          BP = V_SCANDOOR (016EH)

On Exit:  AH = Return status
          BS = New last used DS

Registers Altered: AX, BX, BP, DS
```

SF__START (AX = 0202H)

This subfunction starts the driver.

CAUTION

This function should not be called directly by an application program.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BP = V_SCANDOOR (016EH)

On Exit:  AH = Return status

Registers Altered: AX, BP, DS
```

SF_VERSION (AX = 0206H)

This subfunction will return the release code and a pointer to the current version number.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_SCANDOOR (016EH)

On Exit:  AH = Return status
          BX = Release code
          CX = Number of bytes in version number
          ES:DI = Address of current version number string

Registers Altered: AX, BX, BP, CX, DI, BP, ES, DS
```

SF_GET_STATE (AX = 0800H)

This subfunction will return one of the STATE bytes maintained by the V_SCANDOOR driver. The STATE byte number requested is passed to the driver in BL.

```
On Entry: AH = F_STATE_IOCTL (08H)
          AL = SF_GET_STATE (00H)
          BL = State byte number
          BP = V_SCANDOOR (016EH)

On Exit:  AH = Return status
          BH = STATE bits
```

State Byte	Bit	Meaning
0	0	Reserved
	1	Reserved
	2	SCAN_DOOR_OPEN
	3	SVC_DOOR_OPEN
1	0	BEEP_ENABLE
	1	SPEED_PARSE_ENABLE
	2	CLICK_ENABLE
	3	CLICK_PARSE_ENABLE
2	0	SCANDOR_STATE_INTS_ON
	1	SCANDOR_CERB_INTS_ON
	2	SCANDOR_SCAN_INTS_ON
	3	SCANDOR_SVC_INTS_ON
3	0	Reserved
	1	LOW_CPU_SPEED
	2	Reserved
	3	Reserved

```
Registers Altered: AX, BX, BP, DS
```

(In the above, "CERB" refers to the HP-HIL controller.)

5

Keyboard

Overview

The Keyboard Input System for two keyboards are discussed in this chapter:

- The **HP Vectra Keyboard/DIN** (shown in Figure 5-2) which is used with the HP Vectra ES series of personal computers.
- The **HP Vectra Enhanced Keyboard** (shown in Figure 5-3) which is used with the HP Vectra *series* of personal computers.

Information presented in this chapter will apply to both keyboards except when specified as **Keyboard/DIN only** (for the HP Vectra keyboard/DIN) or **Enhanced keyboard only** (for the HP Vectra Enhanced keyboard).

The Keyboard Input System consists of four components: the input device drivers, STD-BIOS keyboard drivers, 8042 keyboard controller chip and the EX-BIOS keyboard drivers (see Figure 5-1). The input device drivers are discussed in Chapter 4. The other three components are discussed in this chapter.

The industry standard INT 16H and INT 09H handlers make up the STD-BIOS keyboard drivers. INT 16H is used by applications to get characters from the keyboard buffer. INT 09H responds to interrupts from the 8042 controller and places characters in the keyboard buffer.

The 8042 controller chip provides an industry standard hardware interface to the INT 09H driver. It also provides timers and other services to the Input System.

The EX-BIOS drivers allow applications to redefine the scancodes generated by certain groups of keys on the HP Vectra **Keyboard/DIN only**.

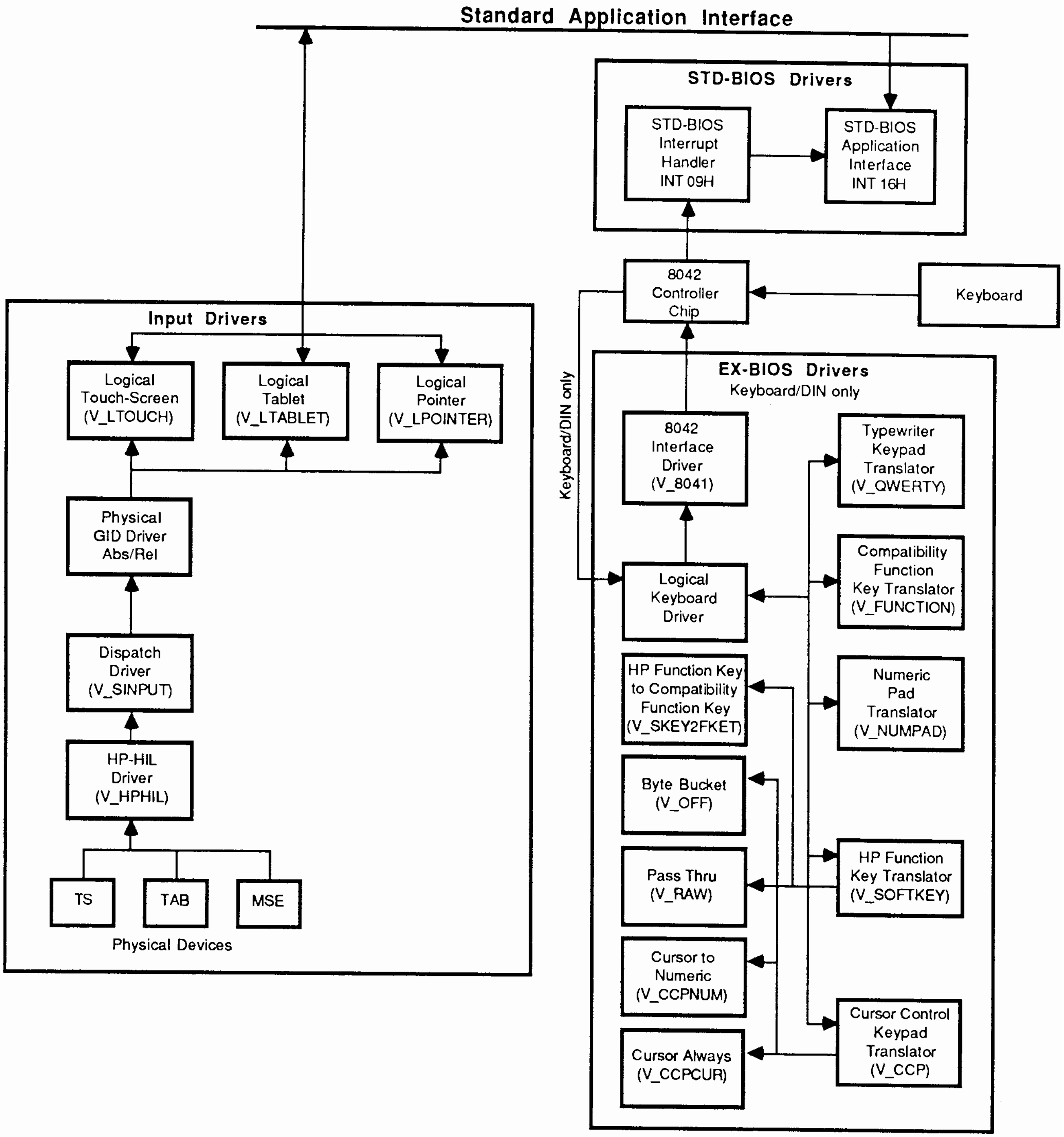


Figure 5-1. Keyboard Block Diagram

Keyboard Drivers

The STD-BIOS component consists of two drivers: the keyboard ISR routine (INT 09H), and the keyboard interface driver (INT 16H).

Overview

The INT 09H driver responds to the 8042 OBF interrupt and reads in a scancode from the 8042 controller. If the scancode is from one of the keyboard modifier keys, the appropriate state bits are updated. The scancode is then placed in the STD-BIOS keyboard buffer along with its corresponding ASCII character (keycode) or a null byte (0H).

The INT 16H driver provides functions to allow the application to interrogate and manipulate the keyboard input system. Applications may check for keycodes in the STD-BIOS keyboard buffer, remove keycodes from it, retrieve the state of the keyboard modifiers, and put keycodes into the STD-BIOS keyboard buffer. Applications may also inquire about and/or change the typematic rate and delay values for the keyboard.

Extended functions (supported with the keyboard/DIN only) are provided by the INT 16H driver to give the application additional control over the keyboard and to facilitate keyboard driver mapping. Extended functions allow the application to turn off or change the default translations performed on the HP Function keypads and Cursor Control keypads (see Figure 5-2). Functions are also provided to aid applications that install keypad translator services of their own.

Data Structures

The INT 16H and INT 09H driver data structures are located in the STD-BIOS data area. They are stored in memory addresses 417H (40:17H) through 43DH (40:3DH), 496H (40:96H) and 497H (40:97H). Table 5-1 lists these memory locations and their definitions.

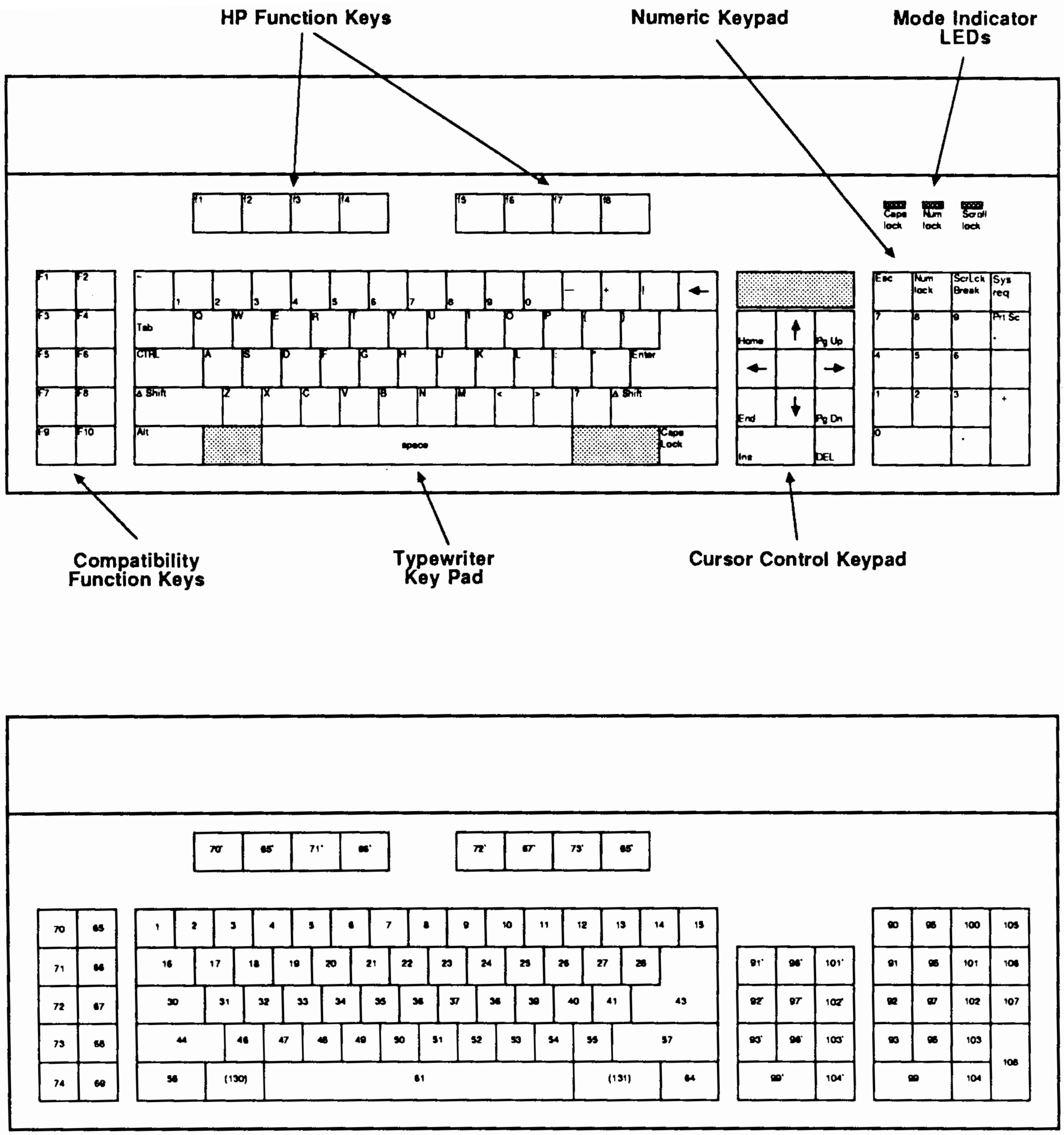


Figure 5-2. HP Vectra Keyboard/DIN Keyboard Layout

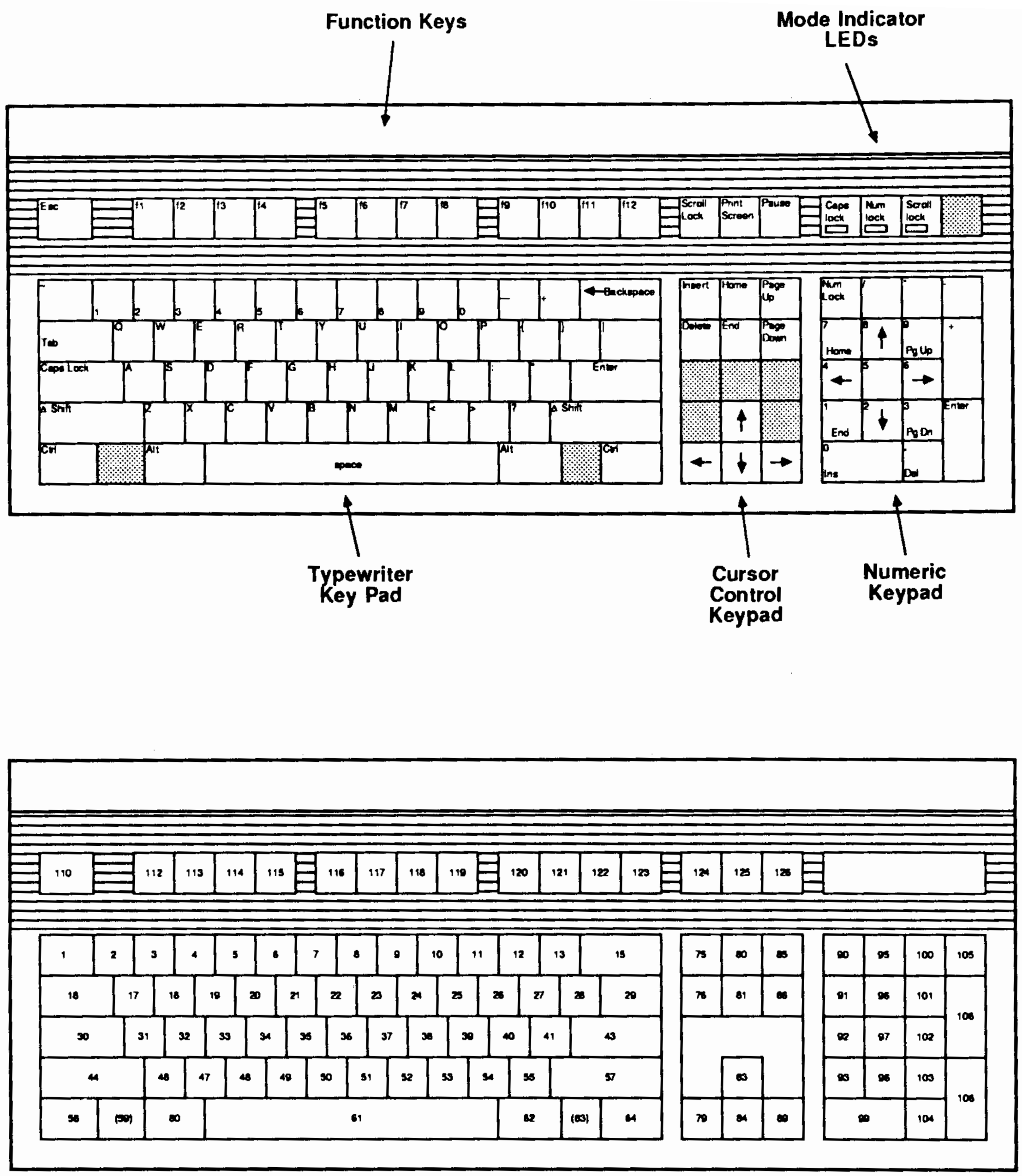


Figure 5-3. HP Vectra Enhanced Keyboard Layout

Table 5-1. STD-BIOS Keyboard Driver Data Area

Address	Length Bytes	Definition
00417H	2	Keyboard Flags
00419H	1	Alt/Numpad accumulator
0041AH	2	Keyboard buffer head pointer
0041CH	2	Keyboard buffer tail pointer
0041EH	32	Keyboard buffer
00496H	1	Extended keyboard flags
00497H	1	Keyboard LED and data flags

The keyboard buffer can store up to 16 entries. Each buffer entry consists of two bytes: an ASCII character (keycode) and a scancode. The keycode and the scancode are placed in the keyboard buffer by the INT 09H driver, and the keyboard head pointer is adjusted accordingly. They are retrieved from the buffer by the INT 16H driver, and the keyboard tail pointer is adjusted.

The keyboard flags are maintained by the INT 09H driver. These flags indicate the state of the keyboard modifier keys and their respective modes. The byte at memory location 417H indicates the mode, the byte at 418H reflects the actual state of the keys themselves, the byte at 496H indicates the state of the extended keyboard processing, and the byte at 497H gives keyboard LED status and data received from the keyboard. Tables 5-2 through 5-5 list these flags and their meaning.

Table 5-2. Keyboard Shift Flags (Address 417H)

Bit	Data	Definition
07H		<Ins> key state
	1	Insert mode is active
06H		<Caps lock> key state
	1	Caps lock mode is active
05H		<Num lock> key state
	1	Num lock mode is active
04H		<Scroll lock> key state
	1	Scroll lock mode is active
03H		<Alt> key state
	1	<Alt> key is pressed

Table 5-2. Keyboard Shift Flags (Address 417H) (Cont.)

Bit	Data	Definition
02H		<Ctrl> key state
	1	<Ctrl> key is pressed
01H		Left <Shift> key state
	1	Left <Shift> key is pressed
00H		Right <Shift> key state
	1	Right <Shift> key pressed

Table 5-3. Keyboard Secondary Shift Flags (Address 418H)

Bit	Data	Definition
07H		<Ins> key state
	1	<Ins> key is pressed
06H		<Caps lock> key state
	1	<Caps lock> key is pressed
05H		<Num lock> key state
	1	<Num lock> key is pressed
04H		<Scroll lock> key state
	1	<Scroll lock> key is pressed
03H		Pause State
	1	Indicates the <Ctrl>-<Num lock> pause state is active
02H		<System request> key state
	1	<System request> key is pressed
01H		Left <Alt> key state
	1	Left <Alt> key is pressed
00H		Left <Ctrl> key state
	1	Left <Ctrl> key is pressed

Table 5-4. 101-key Keyboard Flags (Address 496H)

Bit	Data	Definition
07H	1	Read ID bytes in progress
06H	1	First of ID bytes was last
05H	1	Force Num Lock if 101-key keyboard is attached. This is when DOS is loaded or reloaded. Enhanced Keyboard only
04H	1	101-key keyboard attached. Enhanced Keyboard only
03H		Right <Alt> key status
	1	Right <Alt> key is pressed
02H		Right <Ctrl> key status
	1	Right <Ctrl> key is pressed
01H	1	E0 was last
00H	1	E1 was last

Table 5-5. Keyboard LED and Flags Data Area (Address 497H)

Bit	Data	Definition
07H	1	Used for a flag to indicate 3 failures of sending data to keyboard
06H	1	LED update in progress
05H	1	Resend received from keyboard
04H	1	Acknowledge received from keyboard
03H	0	Reserved (set to 0)
02H		Caps Lock LED status
	1	Caps Lock LED on
01H		Num Lock LED status
	1	Num Lock LED on
00H		Scroll Lock LED status
	1	Scroll Lock LED on

Note: Applications which modify these bytes may experience difficulty in maintaining synchronization between the Cursor Control keypad and the Numeric keypad on the HP Vectra Keyboard/DIN only.

STD-BIOS Keyboard ISR (INT 09H)

The keyboard interrupt service routine is responsible for retrieving scancodes from the 8042 controller, generating the associated keycodes, and placing them into the STD-BIOS keyboard buffer. Certain keys and key combinations do not generate a standard ASCII character code. In these cases a keycode equal to 0 indicates that an application program should examine the scancode byte to determine the "extended" ASCII code. Tables 5-6a through 5-6c contains the scancode to keycode translation assignments.

TABLE 5-6a. SCANCODE TRANSLATION TABLE

Enhancd. keybd. Key Number	Keybd/DIN keybd. Key Number	AT Scancode	HP Scancode	Key Cap	Unshifted ASCII	Unshifted Hex	Shifted ASCII	Shifted Hex	Ctrl	Alt
110	90	76H	01H	Esc	esc	1BH	esc	1BH	1BH	00/01H
02	02	16H	02H	1	1	31H	!	21H	–	00/78H
03	03	1EH	03H	2	2	32H	@	40H	00H	00/79H
04	04	26H	04H	3	3	33H	#	23H	–	00/7AH
05	05	25H	05H	4	4	34H	$	24H	–	00/7BH
06	06	2EH	06H	5	5	35H	%	25H	–	00/7CH
07	07	36H	07H	6	6	36H	^	5EH	1EH	00/7DH
08	08	3DH	08H	7	7	37H	&	26H	–	00/7EH
09	09	3EH	09H	8	8	38H	*	2AH	–	00/7FH
10	10	46H	0AH	9	9	39H	(	28H	–	00/80H
11	11	45H	0BH	0	0	30H	)	29H	–	00/81H
12	12	4EH	0CH	-	-	2DH	_	5FH	1FH	00/82H
13	13	55H	0DH	=	=	3DH	+	2BH	–	00/83H
15	15	66H	0EH	BkSp	bs	08H	bs	08H	7FH	00/0EH
16	16	0DH	0FH	Tab	tab	09H	Nul	00H	00/94H	00/A5H
17	17	15H	10H	Q	q	71H	Q	51H	11H	00/10H
18	18	1DH	11H	W	w	77H	W	57H	17H	00/11H
19	19	24H	12H	E	e	65H	E	45H	05H	00/12H
20	20	2DH	13H	R	r	72H	R	52H	12H	00/13H
21	21	2CH	14H	T	t	74H	T	54H	14H	00/14H
22	22	35H	15H	Y	y	79H	Y	59H	19H	00/15H
23	23	3CH	16H	U	u	75H	U	55H	15H	00/16H
24	24	43H	17H	I	i	69H	I	49H	09H	00/17H
25	25	44H	18H	O	o	6FH	O	4FH	0FH	00/18H
26	26	4DH	19H	P	p	70H	P	50H	10H	00/19H
27	27	54H	1AH	[	[	5BH	{	7BH	1BH	00/1AH
28	28	5BH	1BH	]	]	5DH	}	7DH	1DH	00/1BH
43	43	5AH	1CH	Enter	cr	0DH	cr	0DH	0AH	00/1CH
58	30	14H	1DH	CTRL	–	–	–	–	–	–
31	31	1CH	1EH	A	a	61H	A	41H	01H	00/1EH
32	32	1BH	1FH	S	s	73H	S	53H	13H	00/1FH
33	33	23H	20H	D	d	64H	D	44H	04H	00/20H
34	34	2BH	21H	F	f	66H	F	46H	06H	00/21H
35	35	34H	22H	G	g	67H	G	47H	07H	00/22H
36	36	33H	23H	H	h	68H	H	48H	08H	00/23H
37	37	3BH	24H	J	j	6AH	J	4AH	0AH	00/24H
38	38	42H	25H	K	k	6BH	K	4BH	0BH	00/25H
39	39	4BH	26H	L	l	6CH	L	4CH	0CH	00/26H
40	40	4CH	27H	;	;	3BH	:	3AH	–	00/27H
41	41	52H	28H	'	'	27H	"	22H	–	00/28H
01	01	0EH	29H	`	`	60H	~	7EH	–	00/29H
44	44	12H	2AH	L Shift	–	–	–	–	–	–
29	14	5DH	2BH	\	\	5CH	|	7CH	1CH	00/2BH
46	46	1AH	2CH	Z	z	7AH	Z	5AH	1AH	00/2CH
47	47	22H	2DH	X	x	78H	X	58H	18H	00/2DH
48	48	21H	2EH	C	c	63H	C	43H	03H	00/2EH
49	49	2AH	2FH	V	v	76H	V	56H	16H	00/2FH
50	50	32H	30H	B	b	62H	B	42H	02H	00/30H
51	51	31H	31H	N	n	6EH	N	4EH	0EH	00/31H
52	52	3AH	32H	M	m	6DH	M	4DH	0DH	00/32H
53	53	41H	33H	,	,	2CH	<	3CH	–	00/33H
54	54	49H	34H	.	.	2EH	>	3EH	–	00/34H
55	55	4AH	35H	/	/	2FH	?	3FH	–	00/35H
57	57	59H	36H	R Shift	–	–	–	–	–	–
60	58	11H	38H	Alt	–	–	–	–	–	–
61	61	29H	39H	Space	sp	20H	sp	20H	20H	20H
30	64	58H	3AH	Caps lock	–	–	–	–	–	–

(Cont.)

TABLE 5-6a. SCANCODE TRANSLATION TABLE (cont.)

Enhancd. keybd. Key Number	Keybd/DIN keybd. Key Number	AT Scancode	HP Scancode	Key Cap	Unshifted ASCII	Unshifted Hex	Shifted ASCII	Shifted Hex	Ctrl	Alt
112	70	05H	3BH	F1	–	00/3BH	–	00/54H	00/5EH	00/68H
113	65	06H	3CH	F2	–	00/3CH	–	00/55H	00/5FH	00/69H
114	71	04H	3DH	F3	–	00/3DH	–	00/56H	00/60H	00/6AH
115	66	0CH	3EH	F4	–	00/3EH	–	00/57H	00/61H	00/6BH
116	72	03H	3FH	F5	–	00/3FH	–	00/58H	00/62H	00/6CH
117	67	0BH	40H	F6	–	00/40H	–	00/59H	00/63H	00/6DH
118	73	83H	41H	F7	–	00/41H	–	00/5AH	00/64H	00/6EH
119	68	0AH	42H	F8	–	00/42H	–	00/5BH	00/65H	00/6FH
120	74	01H	43H	F9	–	00/43H	–	00/5CH	00/66H	00/70H
121	69	09H	44H	F10	–	00/44H	–	00/5DH	00/67H	00/71H

Enhancd. keybd. Key Number	Keybd/DIN keybd. Key Number	AT Scancode	HP Scancode	Key Cap	NumLock or Shift ASCII	NumLock or Shift Hex	NumLock and Shift	None or CTRL	Alt
90	95	77H	45H	Num Lock	–	45H	–	–	–
125	100	7EH	46H	ScrLck	–	46H	–	00/00H	–
91	91	6CH	47H	Home	7	37H	00/47H	00/77H	–
96	96	75H	48H		8	38H	00/48H	00/8DH	–
101	101	7DH	49H	Pg Up	9	39H	00/49H	00/84H	–
105	107	7BH	4AH	-	-	2DH	2DH	00/8EH	00/4AH
92	92	6BH	4BH		4	34H	00/4BH	00/73H	–
97	97	73H	4CH	5	5	35H	00/4CH	00/8FH	–
102	102	74H	4DH		6	36H	00/4DH	00/74H	–
106	108	79H	4EH	+	+	2BH	2BH	00/90H	00/4EH
93	93	69H	4FH	End	1	31H	00/4FH	00/75H	–
98	98	72H	50H		2	32H	00/50H	00/91H	–
103	103	7AH	51H	Pg Dn	3	33H	00/51H	00/76H	–
99	99	70H	52H	Ins	0	30H	00/52H	00/92H	–
104	104	71H	53H	DEL	.	2EH	00/53H	00/93H	–

TABLE 5-6b. KEYBOARD/DIN KEYBOARD SPECIFIC KEYS

Key Number	AT Scancode	HP Scancode	Key Cap	Unshifted	Shifted	CTRL	Alt
106	7CH	37H	* (NmPd)	2AH	(Prt Sc)	00/72H	00/37H
105	84H	54H	Sysreq	–	–	–	–
–		55H	– undef.				
–		56H	– undef.				
–		57H	– undef.				
–		58H	– undef.				
–		59H	– undef.				
–		5AH	– undef.				
–		5BH	– undef.				
–		5CH	– undef.				
–		5DH	– undef.				
59		5EH	Unlabeled-L	00/D7H	00/BDH	00/A3H	00/89H
62		5FH	Unlabeled-R	00/D8H	00/BEH	00/A4H	00/8AH
113		60H	CCP-Up	00/D9H	00/BFH	00/A5H	00/8BH
111		61H	CCP-Lft	00/DAH	00/C0H	00/A6H	00/8CH
115		62H	CCP-Dn	00/DBH	00/C1H	00/A7H	00/8DH
118		63H	CCP-Rht	00/DCH	00/C2H	00/A8H	00/8EH
110		64H	CCP-Home	00/DDH	00/C3H	00/A9H	00/8FH
117		65H	CCP-PgUp	00/DEH	00/C4H	00/AAH	00/90H
112		66H	CCP-End	00/DFH	00/C5H	00/ABH	00/91H
119		67H	CCP-PgDn	00/E0H	00/C6H	00/ACH	00/92H
116		68H	CCP-Ins	00/E1H	00/C7H	00/ADH	00/93H
120		69H	CCP-Del	00/E2H	00/C8H	00/AEH	00/94H
114		6AH	CCP-CNTR	00/E3H	00/C9H	00/AFH	00/95H
		6BH	– undef.	00/E4H	00/CAH	00/B0H	00/96H
		6CH	– undef.	00/E5H	00/CBH	00/B1H	00/97H
		6DH	– undef.	00/E6H	00/CCH	00/B2H	00/98H
		6EH	– undef.	00/E7H	00/CDH	00/B3H	00/99H
		6FH	– undef.	00/E8H	00/CEH	00/B4H	00/9AH
121		70H	f1	00/E9H	00/CFH	00/B5H	00/9BH
122		71H	f2	00/EAH	00/D0H	00/B6H	00/9CH
123		72H	f3	00/EBH	00/D1H	00/B7H	00/9DH
124		73H	f4	00/ECH	00/D2H	00/B8H	00/9EH
125		74H	f5	00/EDH	00/D3H	00/B9H	00/9FH
126		75H	f6	00/EEH	00/D4H	00/BAH	00/A0H
127		76H	f7	00/EFH	00/D5H	00/BBH	00/A1H
128		77H	f8	00/F0H	00/D6H	00/BCH	00/A2H
		78H through 7FH	– undef.				

TABLE 5-6c. ENHANCED KEYBOARD SPECIFIC KEYS

Key Number	AT Scancode	HP Scancode	Key Cap	Unshifted ASCII	Unshifted Hex	Shifted ASCII	Shifted Hex	CTRL	Alt
100	7CH	37H	* (NmPd)	–	2AH	–	2AH	00/96H	00/37H
122	78H	57H	F11	–	00/85H	–	00/87H	00/89H	00/8BH
123	07H	58H	F12	–	00/86H	–	00/88H	00/8AH	00/8CH
124 w/o Alt	E0H, 12H, E0H, 7CH	E0H, 2AH, E0H, 37H	PrtScrn (w/o Alt)	–	–	–	–	00/72H	–
124 w/ Alt	84H	54H	SysReq (w/ Alt)	–	–	–	–	–	–
126 w/o Cntrl	E1H, 14H, 77H, E1H, F0H, 14H, F0H, 77H	E1H, 1DH, 45H, E1H, 9DH, C5H	Pause (w/o Cntrl)	–	–	–	–	–	–
126 w/ Cntrl	E0H, 7EH, E0H, F0H, 7EH	E0H, 46H, E0H, C6H	Break (w/ Cntrl)	–	–	–	–	–	–
Duplicate Keys									
64	E0H, 14H	E0H, 1DH	R Cntrl	–	–	–	–	–	–
108	E0H, 5AH	E0H, 1CH	NmPd Ent	–	0D/E0H	–	0D/E0H	0A/E0H	00/A6H
95	E0H, 4AH	E0H, 35H	NmPd /	–	2F/E0H	–	2F/E0H	00/95H	00/A4H
62	E0H, 11H	E0H, 38H	Rt Alt	–	–	–	–	–	–
80 (w/ NmLk)	E0H, 12H, E0H, 6CH	E0H, 2AH, E0H, 47H	Home (w/ NmLk)	–	E0/47H	–	E0/47H	E0/77H	00/97H
80 (w/o NmLk)	E0H, 6CH,	E0H, 47H	Home (w/o NmLk)	–	E0/47H	–	E0/47H	E0/77H	00/97H
89 (w/ NmLk)	E0H, 12H, E0H, 6BH	E0H, 2AH, E0H, 4BH	Left Arrow (w/ NmLk)	–	E0/4BH	–	E0/4BH	E0/73H	00/9BH
89 (w/o NmLk)	E0H, 6BH,	E0H, 4BH,	Left Arrow (w/o NmLk)	–	E0/4BH	–	E0/4BH	E0/73H	00/9BH
81 (w/ NmLk)	E0H, 12H, E0H, 69H	E0H, 2AH, E0H, 4FH	End (w/ NmLk)	-	E0/4FH	-	E0/4FH	E0/75H	00/9FH
81 (w/o NmLk)	E0H, 69H,	E0H, 4FH,	End (w/o NmLk)	-	E0/4FH	-	E0/4FH	E0/75H	00/9FH
83 (w/ NmLk)	E0H, 12H, E0H, 75H	E0H, 2AH, E0H, 48H	Up Arrow (w/ NmLk)	-	E0/48H	-	E0/48H	E0/8DH	00/98H
83 (w/o NmLk)	E0H, 75H,	E0H, 48H,	Up Arrow (w/o NmLk)	-	E0/48H	-	E0/48H	E0/8DH	00/98H
84 (w/ NmLk)	E0H, 12H, E0H, 72H	E0H, 2AH, E0H, 50H	Dwn Arrow (w/ NmLk)	-	E0/50H	-	E0/50H	E0/91H	00/A0H
84 (w/o NmLk)	E0H, 72H,	E0H, 50H,	Dwn Arrow (w/o NmLk)	-	E0/50H	-	E0/50H	E0/91H	00/A0H
75 (w/ NmLk)	E0H, 12H, E0H, 70H	E0H, 2AH, E0H, 52H	Insert (w/ NmLk)	-	E0/52H	-	E0/52H	E0/92H	00/A2H
75 (w/o NmLk)	E0H, 70H,	E0H, 52H,	Insert (w/o NmLk)	-	E0/52H	-	E0/52H	E0/92H	00/A2H
85 (w/ NmLk)	E0H, 12H, E0H, 7DH	E0H, 2AH, E0H, 49H	Page Up (w/ NmLk)	-	E0/49H	-	E0/49H	E0/84H	00/99H
85 (w/o NmLk)	E0H, 7DH,	E0H, 49H,	Page Up (w/o NmLk)	-	E0/49H	-	E0/49H	E0/84H	00/99H
89 (w/ NmLk)	E0H, 12H, E0H, 74H	E0H, 2AH, E0H, 4DH	Right Arrow (w/ NmLk)	-	E0/4DH	-	E0/4DH	E0/74H	00/9DH
89 (w/o NmLk)	E0H, 74H,	E0H, 4DH,	Right Arrow (w/o NmLk)	-	E0/4DH	-	E0/4DH	E0/74H	00/9DH
86 (w/ NmLk)	E0H, 12H, E0H, 7AH	E0H, 2AH, E0H, 51H	Page Dwn (w/ NmLk)	-	E0/51H	-	E0/51H	E0/76H	00/A1H
86 (w/o NmLk)	E0H, 7AH,	E0H, 51H,	Page Dwn (w/o NmLk)	-	E0/51H	-	E0/51H	E0/76H	00/A1H
76 (w/ NmLk)	E0H, 12H, E0H, 71H	E0H, 2AH, E0H, 53H	Delete (w/ NmLk)	-	E0/53H	-	E0/53H	E0/93H	00/A3H
76 (w/o NmLk)	E0H, 71H,	E0H, 53H,	Delete (w/o NmLk)	-	E0/53H	-	E0/53H	E0/93H	00/A3H
Hidden Keys									
14	6AH	7DH	–	–	–	–	–	–	–
42	5DH	2BH	–	–	5CH	–	7CH	1C/2BH	00/2BH
45	61H	56H	–	–	5CH	–	7CH	7C/56H	7C/56H
56	51H	73H	–	–	–	–	–	–	–
59	–	5EH	–	–	00/D7H	–	00/BDH	00/A3H	00/89H
63	–	5FH	–	–	00/D8H	–	00/BEH	00/A4H	00/8AH
94	68H	7CH	–	–	–	–	–	–	–
107	6DH	7EH	–	–	–	–	–	–	–
109	63H	78H	–	–	–	–	–	–	–

The INT 09H driver tracks the state of the keyboard modifiers presented in Tables 5-2 through 5-5 as well as processing the special key sequences in Table 5-7.

Table 5-7. INT 09H Special Key Sequences

Key Combinations	Action
<Pause>	Stops execution until any non-shift key on the keyboard is struck. Enhanced Keyboard only
<Ctrl>-<Num lock>	Stops execution until any non-shift key on the keyboard is struck. Keyboard/DIN only
<Ctrl>-<Alt>-<+>	This key sequence enables the key click feature. The longer the these keys are held down together, the louder the key click. After maximum volume is achieved the key click volume will wrap around to low volume. This is done in the 8042
<Ctrl>-<Alt>-<->	This key sequence reduces the key click volume until it is off. This is done in the 8042.
<Ctrl>-<Alt>-<\>	This key sequence toggles the computer speed. (On the Vectra ES, this is handled by the 8042. On the Vectra QS and RS this is handled by the system BIOS.)
<Ctrl>-<Break>	This key sequence is interpreted as a program break request. When this key sequence is detected, the INT 09H driver will execute an INT 1BH instruction. The vector for this interrupt is initialized during the boot process to point to a routine within MS-DOS which sets a flag then performs an IRET instruction. This vector may be modified to point to an alternate routine to handle a <Ctrl>-<Break>.
<Ctrl>-<Alt>-<DEL>	This key sequence is interpreted as a system reset command. When this key sequence is detected, control is transferred to the BIOS Reset routine.
<Print Screen>	This key is interpreted as a print screen command. When this key is detected, an INT 05H instruction is executed. Enhanced Keyboard only
<Shift>-<Print Screen>	This key sequence is interpreted as a print screen command. When this key sequence is detected, an INT 05H instruction is executed. Keyboard/DIN only
<System request>	This key is interpreted as a system request for multi-tasking.
<Alt>-nnn	Where nnn represents a three digit decimal number entered on the numeric keypad which yields the associated ASCII characters, i.e., <Alt>-122 yields the character "z".

STD-BIOS Keyboard Driver (INT 16H)

The INT 16H driver acts as the interface between applications and the keyboard. This driver has two sets of functions. One set provides functions to return keycodes and keyboard status. The other set of functions allows the application to change the translation algorithms of the scancodes and to vary the repeat rates of the keys on the Keyboard/DIN only. Table 5-8 is a summary of this driver's function codes.

Table 5-8. Keyboard Driver (INT 16H) Function Code Summary

Function Equate	Function Definition	Hex Value
INT_KBD	Keyboard	
F16_GET_KEY	Read keycode from keyboard buffer	00H
F16_STATUS	Report Status of keyboard buffer	01H
F16_KEY_STATE	Get Key Modifier Status	02H
F16_SET _TYPE_RATE	Set typematic rates	03H
F16_PUT_KEY	Put data into keyboard buffer	05H
F16_GET_EXT_KEY	Read keycode from buffer (including new Vectra ES, QS, and RS keycodes)	10H
F16_EXT_STATUS	Report keyboard status (including new Vectra ES, QS, and RS keycodes)	11H
F16_EXT _KEY_STATE	Get extended key modifier status	12H
F16_INQUIRE	EX-BIOS present	6F00H
F16_DEF_ATTR	Report default values for repeat rates and delay time before repeat Keyboard/DIN only	6F01H
F16_GET_ATTR	Report current repeat rates and delay time Keyboard/DIN only	6F02H
F16_SET_ATTR	Replaces current repeat rates and delay time Keyboard/DIN only	6F03H
F16_DEF _MAPPING	Reports default HP-system vector entries for keyboard translator drivers Keyboard/DIN only	6F04H

Table 5-8. Keyboard Driver (INT 16H) Function Code Summary (Cont.)

Function Equate	Function Definition	Hex Value
F16_GET _MAPPING	Reports current HP-system vector entries for keyboard translator drivers Keyboard/DIN only	6F05H
F16_SET _MAPPING	Replaces current HP-system vector entries for keyboard translator drivers Keyboard/DIN only	6F06H
F16_SET _XLATORS	Switches either the cursor controlpad translator or the HP Function keypad translator functions of the keyboard Keyboard/DIN only	6F07H
F16_KBD	Reports keyboard identification Keyboard/DIN only	6F08H
F16_KBD_RESET	Reset logical keyboard structure to defaults Keyboard/DIN only	6F09H
F16_READ_SPEED	Read current speed	6F0AH
F16_SET_LOW _SPEED	Selects the low speed for the computer	6F0BH
F16_SET_HIGH _SPEED	Selects the high speed for the computer	6F0CH
F16_GET_INT _NUMBER	Returns the current HPENTRY interrupt number	6F0DH

Keyboard Driver (INT 16H) Function Definitions

F16_GET_KEY (AH = 00H)

This function returns the next keycode from the keyboard buffer. If no keycode is ready, this function waits for one. This function **does not** return all keycodes available on the HP Vectra series of computers. It returns those keycodes that are available on the original HP Vectra PC. The new keycodes are thrown away.

```
On Entry: AH = F16_GET_KEY (00H)

On Exit: AH = Scancode
         AL = ASCII keycode or extended keycode

Registers Altered: AX
```

F16_STATUS (AH = 01H)

This function returns the status of the keyboard buffer. The Zero flag is cleared if a keycode is available, or set if there is no keycode in the buffer. If a keycode is ready, the scancode and keycode are returned in the AH and AL registers respectively. Even though the scancode and keycode are returned with this function, they must be read with F16_GET_KEY to remove them from the keyboard buffer. This function **does not** return all keycodes available on the HP Vectra series of computers. It returns those keycodes that are available on the original HP Vectra PC. The new keycodes are thrown away.

```
On Entry: AH = F16_STATUS (01H)

On Exit: Z = 1 if no keycode is ready.
         Z = 0 if a keycode is ready.

          and

        AH = Scancode
        AL = Keycode or extended keycode.

Registers Altered: AX
```

F16_KEY_STATE (AH = 02H)

This function returns the state of the various keyboard modifiers that were availble on the original HP Vectra PC. The status byte returned is a copy of the keyboard modifier status byte stored at memory location 417H.

On Entry: AH = F16_KEY_STATE (02H)

On Exit: AL = Modifier Status Byte

Bit	Data	Definition
07H	1	Insert mode active
	0	Insert mode inactive
06H	1	Caps lock mode active
	0	Caps lock mode inactive
05H	1	Num lock mode active
	0	Num lock mode inactive
04H	1	Scroll lock mode active
	0	Scroll lock mode inactive
03H	1	<Alt> key pressed
	0	<Alt> key released
02H	1	<Ctrl> key pressed
	0	<Ctrl> key released
01H	1	Left <Shift> key pressed
	0	Left <Shift> key released
00H	1	Right <Shift> key pressed
	0	Right <Shift> key released

Registers Altered: AL

F16_SET_TYPE_RATE (AH = 03H)

This command sets the values for the typematic rate and delay. The typematic rate is the number of make scancodes per second sent in the typematic (repeat) mode. The delay is the amount of time a key must be held down until it enters the typematic mode.

On Entry: AH = F16_SET_TYPE_RATE (03H)
AL = 05
BH = Typematic Delay (00-03H)
BL = Typematic Rate (00-1FH)

On Exit: None

Registers Altered: AX

F16_PUT_KEY (AH = 05H)

This command puts a scancode and a keycode in the keyboard buffer. When this is done, it looks just like INT 9 placed the scancode and keycode there. It may be read with INT 16 functions 0, 1, 10 and 11.

```
On Entry: AH = F16_PUT_KEY (05H)
          CX = Data to place in keyboard buffer
               CH = Scancode
               CL = Keycode or extended keycode

On Exit:  AL = 00 if store successful AL = 01 if not

Registers Altered: AX
```

F16_GET_EXT_KEY (AH = 10H)

This function returns the next keycode fromthe keyboard buffer. If no keycode is ready, this function waits for one. All keycodes are returned; none are thrown away

```
On Entry: AH = F_16_GET_EXT_KEY (10H)

On Exit:  AH = Scancode
          AL = Keycode or extended keycode

Registers Altered: AX
```

F16_EXT_STATUS (AH = 11H)

This function returns the status of the keyboard buffer. The Zero flag is cleared if a keycode is available, or set if there is no keycode in the buffer. If a keycode is ready, the scacode and keycode are returned in the AH and AL registers respectively. Enen though the scancode and keycode are returned with this function, they must be read with F16_GET_EXT_KEY to remove them from the keyboard buffer. All keycodes are returned; none are thrown away.

```
On Entry: AH = F16_EXT_STATUS (11H)

On Exit:  AH = Scancode
          AL = Keycode or extended keycode
               Z = 1 if no keycode is ready
               Z = 0 if a keycode is ready

Registers Altered: AX, flag
```

F16_EXT_KEY_STATE (AH = 12H)

This function returns the state of various keyboard modifiers, including the new states available on the HP Vectra series of computers. AL is a copy of the keyboard modifier status byte stored at memory location 417H. AH is a combination of some of the bits stored in memory location 418H and 496H.

On Entry: AH = F16_EXT_KEY_STATE (12H)

On Exit: AH = Extended Modifier Status
AL = Modifier Status Byte

AL:

Bit	Data	Definition
07H	1	Insert mode active
	0	Insert mode inactive
06H	1	Caps lock mode active
	0	Caps lock mode inactive
05H	1	Num lock mode active
	0	Num lock mode inactive
04H	1	Scroll lock mode active
	0	Scroll lock mode inactive
03H	1	<Alt> key pressed
	0	<Alt> key released
02H	1	<Ctrl> key pressed
	0	<Ctrl> key released
01H	1	Left <Shift> key pressed
	0	Left <Shift> key released
00H	1	Right <Shift> key pressed
	0	Right <Shift> key released

AH:

Bit	Data	Definition	Concatenated From
07H	1	<System request> key pressed	bit 2 418H
	0	<System request> key released	
06H	1	<Caps lock> key pressed	bit 6 418H
	0	<Caps lock> key released	
05H	1	<Num lock> key pressed	bit 5 418H
	0	<Num lock> key released	
04H	1	<Scroll lock> key pressed	bit 4 418H
	0	<Scroll lock> key released	
03H	1	Right <Alt> key pressed	bit 3 496H
	0	Right <Alt> key released	
02H	1	Right <Ctrl> key pressed	bit 2 496H
	0	Right <Ctrl> key released	
01H	1	Left <Alt> key pressed	bit 1 418H
	0	Left <Alt> key released	
00H	1	Left <Ctrl> key pressed	bit 0 418H
	0	Left <Ctrl> key released	

Registers Altered: AX

F16_INQUIRE (AX = 6F00H)

This subfunction determines whether or not the extended HP functions are available. If the HP functions are available, the BX register will be set to 4850H (which is the ASCII characters 'HP').

```
On Entry: AX = F16_INQUIRE (6F00H)
          BX = Any value except 4850H, 'HP'.

On Exit: BX = 'HP'

Registers Altered: BX
```

F16_DEF_ATTR (AX = 6F01H)

Keyboard/DIN only

This subfunction reports the default typematic rate and delay values for the keyboard. A pointer to a four-byte buffer is returned, but the last 2 bytes in that buffer are ignored. The bytes in the buffer are defined in Table 5-9.

Table 5-9. INT 16H Typematic Buffer Format

Byte	Function
0	Delay before repeat action starts for all keys.
1	Typematic Repeat rate for all keys.

Table 5-10 summarizes the typematic rate and delay values defined for each data byte accepted in the typematic buffer by the INT 16H driver.

Table 5-10. INT 16H Typematic Rates and Delays

Data Byte	Byte 1 Repeat Rate*	Byte 0 Number of Milli-seconds Delayed**
00H	(30.00)	[0.250]
01H	(30.00)	[0.250]
02H	(20.00)	[0.250]
03H	(15.00)	[0.250]
04H	(12.00)	[0.250]
05H	(10.00)	[0.250]
06H	(9.20)	[0.250]
07H	(7.50)	[0.500]
08H	(6.70)	[0.500]
09H	(6.00)	[0.500]

Table 5-10. INT 16H Typematic Rates and Delays (Cont.)

Data Byte	Byte 1 Repeat Rate*	Byte 0 Number of Milli-seconds Delayed**
0AH	(5.50)	[0.500]
0BH	(5.00)	[0.750]
0CH	(4.60)	[0.750]
0DH	(4.30)	[0.750]
0EH	(4.00)	[0.750]
0FH	(2.00)	[0.750]

* Numbers in parentheses () indicate the approximate number of repeated characters per second.

** Numbers in brackets [] indicate the approximate length of delay prior to the first repeated scancode report.

```
On Entry: AX = F16_DEF_ATTR (6F01H)

On Exit: AH = 00H (Successful operation)
      ES:SI = Pointer to buffer
         CX = 4 (Number of entries in table)

Registers Altered: AX, CX, SI, ES
```

F16__GET__ATTR (AX = 6F02H)

Keyboard/DIN only

This subfunction reports the current typematic rate and delay values for the keyboard. A pointer to a four-byte buffer is returned, but the last two bytes are ignored. The bytes in the buffer are interpreted as shown in Table 5-9 and 5-10.

```
On Entry: AX = F16_GET_ATTR (6F02H)

On Exit: AH = 00H (Successful operation)
      ES:SI = Pointer to buffer
         CX = 4 (Number of entries in table)

Registers Altered: AX, CX, SI, ES
```

F16__SET__ATTR (AX = 6F03H)

Keyboard/DIN only

This subfunction sets the current typematic rate and delay values for the keyboard. A pointer to a two-byte buffer is passed, but the second byte is ignored. The bytes in the buffer are interpreted as shown in Table 5-9 and 5-10. Note that the values passed for the rest of the keyboard are also applied to the Cursor Control keypad.

```
On Entry: AX = F16_SET_ATTR (6F03H)
       ES:SI = Pointer to buffer

On Exit: AH = 00H (Successful operation)

Registers Altered: AX
```

F16__DEF__MAPPING (AX = 6F04H)

Keyboard/DIN only

This subfunction reports the default keyboard translator mappings. A pointer to a buffer of 1EH bytes is supplied by the caller to be filled in by the ROM-BIOS. The table will contain the default HP__VECTOR__TABLE entries for each of the five translator drivers. Each of five entries in the table will contain the IP, CS, and DS for each translator driver.

CAUTION

An application should restore the translator drivers to their original condition upon termination. If an application replaces one of these drivers, the programmer should be aware that the EX-BIOS keyboard driver functions 6F07H may no longer function properly.

The format of the buffer is given in Table 5-11.

Table 5-11. INT 16H Mapping Buffer Format

Offset	Translator
00H	Entry for V__QWERTY driver
06H	Entry for V__SOFTKEY driver
0CH	Entry for V__FUNCTION driver
12H	Entry for V__NUMPAD driver
18H	Entry for V__CCP driver

In the above table, note that QWERTY refers to the typewriter keypad, SOFTKEY refers to the HP Function keypad, FUNCTION refers to the Compatibility Function keypad, NUMPAD refers to the

Numeric keypad, CCP refers to the Cursor Control keypad (the location of keypads on the **Keyboard/DIN** are shown in Figure 5-2.)

```
On Entry: AX = F16_DEF_MAPPING (6F04H)
       ES:SI = Pointer to buffer

On Exit: AH = 00H (Successful)
      ES:SI = Pointer to buffer of 1EH bytes
         CX = 1EH (Size of buffer)

Registers Altered: AX, CX
```


F16__GET__MAPPING (AX = 6F05H)

Keyboard/DIN only

This subfunction reports the current keyboard translator mappings. A pointer to a buffer 1EH bytes in length is supplied by the caller to be filled in by the ROM-BIOS. The buffer will contain the current HP__VECTOR__TABLE entries for each of the five translator drivers (IP, CS, and DS for each driver). The format of the buffer is given in Table 5-11.

```
On Entry: AX = F16_GET_MAPPING (6F05H)
       ES:SI = Pointer to buffer

On Exit: AH = 00H (Successful)
      ES:SI = Pointer to buffer
         CX = 1EH (Size of table)

Registers Altered: AX, CX
```

F16__SET__MAPPING (AX = 6F06H)

Keyboard/DIN only

This subfunction sets the current keyboard translator mappings. A pointer to a buffer containing the entries to be written into the HP__VECTOR__TABLE is passed in. The format of the buffer is given in Table 5-11.

A driver that replaces a scancode translator can expect to handle a Keyboard ISR Event Record (Table 5-13). If the translator wishes to remove the passed in scancode from the scancode stream, it returns a status of RS__DONE. Otherwise, a return status of RS__SUCCESSFUL should be set and an appropriate ISR EVENT record returned. The ISR Event Record will then be passed on to the next driver in the chain. The driver can depend on 20H bytes of stack.

```
On Entry: AX = F16_SET_MAPPING (6F06H)
       ES:SI = Pointer to table.
          CX = 01EH (size of table in bytes)

On Exit: AH = 00H (Successful)

Registers Altered: AX
```

F16_SET_XLATORS (AX = 6F07H)

Keyboard/DIN only

This subfunction sets the current mappings of the HP Function keypad (V_SOFTKEY) and Cursor Control keypad (V_CCP) translators. Note that only one translator may be set with each call to this subfunction. (Figure 5-2 shows the possible mappings for these two HP proprietary keypads.)

```
On Entry: AX = F16_SET_XLATORS (6F07H)
          BL = Translation
```

Data	Definition
00H	Maps V_CCP to V_CCPCUR which forces the Cursor Control keypad to generate Numeric keypad cursor key scancodes, regardless of state of <Num lock>. (Default mapping)
01H	Maps V_CCP to V_CCPNUM which forces the Cursor Control keypad to generate Numeric keypad or cursor key scancodes, depending on state of <Num lock>.
02H	Maps V_CCP to V_OFF which disables the Cursor Control keypad.
03H	Maps V_CCP to V_CCPGID (if installed) which converts Cursor Control keypad data to GID data.
04H	Maps V_CCP to V_RAW which passes Cursor Control keypad scancodes untranslated to the INT 09H driver.
05H	Maps V_SOFTKEY to V_SKEY2FKEY which translates HP Function keypad scancodes into equivalent industry standard Compatibility Function keypad scancodes (default mapping).
06H	Maps V_SOFTKEY to V_RAW which passes HP Function keypad scancodes untranslated to INT 09H driver.
07H	Maps V_SOFTKEY to V_OFF which disables HP Function keypad.

On Exit: AH = 00 (Successful)

Registers Altered: AX

F16_KBD (AX = 6F08H)

This subfunction returns the ID of the keyboard.

```
On Entry: AX = F16_KBD (6F08H)

On Exit: AH = 00H (Successful)
           or 02H (Unsupported) if a non-HP keyboard is attached
         BL = Language of the attached keyboard (see below)

Registers Altered: AX, BX

Keyboard Identification:
```

Register BL	Language	Register BL	Language
00	Reserved	10	Chinese (PRC)
01	Arabic-French	11	Chinese (Taiwan)
02	Kanji	12	Swiss (French ii)
03	Swiss-French	13	Spanish
04	Portugese	14	Swiss (German ii)
05	Arabic	15	Belgian (Flemish)
06	Hebrew	16	Finish
07	Canadian-English	17	United Kingdom
08	Turkish	18	French-Canadian
09	Greek	19	French-German
0A	Thai	1A	Norwegian
0B	Italian	1B	French
0C	Hangul (Korean)	1C	Danish
0D	Dutch	1D	Katakana
0E	Swedish	1E	Latin American Spanish
0F	German	1F	United States-American

```
0FFH non-HP keyboard (IBM AT keyboard and IBM Enhanced keyboard)
All others are reserved.
```

F16_KBD_RESET (AX = 6F09H)

Keyboard/DIN only

This subfunction resets all keyboard mappings to their default translators and resets all keyboard typematic rates and delays to their default values.

```
On Entry: AX = F16_KBD_RESET (6F09H)

On Exit: AH = 00H (Successful)

Registers Altered: AX
```

F16_READ_SPEED (AX = 6F0AH)

This subfunction returns a code for the current speed of the computer. Computer speeds for the Vectra series of computers are shown in Table 5-12.

```
On Entry: AX = F16_READ_SPEED (6F0AH)

On Exit:  AH = 00H (Successful)
          BX = 0BH for low speed (see following table)
               12H for medium speed (see following table)
               0CH for high speed (see following table)

Registers Altered: AX, BX
```

Table 5-12. Speeds for HP Vectra Series of Computers

Vectra	High	Medium	Low
ES	8 MHz	-	8 MHz
ES/12	12 MHz	-	8 MHz
QS/16, RS/16	16 MHz	-	8 MHz
QS/20, RS/20	20 MHz	-	8 MHz
RS/20C	20 MHz	10 MHz	5 MHz
RS/25C	25 MHz	12.5 MHz	5 MHz

F16_SET_LOW_SPEED (AX = 6F0BH)

This subfunction sets the speed of the computer to low.

```
On Entry: AX = F16_SET_LOW_SPEED (6F0BH)

On Exit:  AH = 00H (Successful)

Registers Altered: AX
```

F16_SET_HIGH_SPEED (AX = 6F0CH)

This subfunction sets the speed of the computer to high.

```
On Entry: AX = F16_SET_HIGH_SPEED (6F0CH)

On Exit:  AH = 00H (Successful)

Registers Altered: AX
```

F16_GET_INT_NUMBER (AX = 6F0DH)

In the original HP Vectra PC, the HPENTRY vector is INT 6FH. On the HP Vectra series of computers, the default vector is INT 6FH, but it can be moved to another interrupt by the system. If an application programmer wants to use the HPENTRY interrupt, they should do an INT 16 6F0DH function to get the interrupt number in use.

```
On Entry: AX = F16_GET_INT_NUMBER (6F0DH)
On Exit:  AH = Interrupt Number (except when AH = 2, then the
               interrupt number is 6FH)
Registers Altered: AX
```

F16___SET___CACHE___ON (AX = 6F0FH) -- This subfunction enables memory caching.

```
On Entry:   AX = F16_SET_CACHE_ON (6F0FH)
On Exit:    AH = 00H (Successful)
               = FEH (Cache subsystem is bad)
Registers Altered:  AX
```

F16___SET___CACHE___OFF (AX = 6F10H) -- This subfunction disables memory caching.

```
On Entry:   AX = F16__SET__CACHE__OFF (AX = 6F10H)
On Exit:    AH = 00H (Successful)
Registers Altered:  AX
```

F16___GET___CACHE___STATE (AX = 6F11H)

This subfunction returns the memory cache subsystem's state.

```
On Entry:   AX = F16__GET__CACHE__STATE (AX = 6F11H)
On Exit:    AH = 00H (Successful)
            AL bit 0 = 0 (Cache Disabled)
                     = 1 (Cache Enabled)
Registers Altered:  AX
```

F16___SET___MEDIUM___SPEED (AX = 6F12H)

This subfunction sets the computer's speed to medium.

```
On Entry:   AX = F16__SET__MEDIUM__SPEED (6F12H)
On Exit:    AH = 00H (Successful)
Registers Altered:  AX
```

Keyboard Layout Identification

Applications often need to know the layout of the keyboard attached to the system. The following is the recommended algorithm:

1. Check bit 4 in byte 496H. If the bit is one, the keyboard is a HP Vectra **Enhanced keyboard** layout, or an industry-standard 101-key keyboard layout. If the bit is zero, the keyboard is an HP Vectra **Keyboard/DIN** layout, or an industry-standard 84-key keyboard layout.
2. If bit 4 above equals zero, use function 6F00 to determine if the extended functions are present. If not, assume that the keyboard is a non-HP, 84-key keyboard layout.
3. If extended functions are present, use function 6F08 to determine whether the keyboard is an HP Vectra keyboard or some other third party keyboard.

EX-BIOS Keyboard Drivers for the HP Vectra Keyboard/DIN

Keyboard/DIN only

This section discusses **Vectra Keyboard/DIN** information related to ISR events and ISR Event Records, device driver chains, and HP-HIL device data input; these concepts were introduced in Chapter 4.

Overview

The following applies to the Keyboard/DIN only - and only when an INT 16H 6F06 and 6F07 function has been called *or* when one of these functions is called directly.

The EX-BIOS keyboard component consists of the logical keyboard driver, the keyboard translator services, and the V_8042 interface driver.

Logical Keyboard Driver

The logical keyboard driver is the primary interface for the physical keyboard and controls the process of scancode translation. Based on the keypad, the scancode is passed to one of five translator services: V_QWERTY, V_SOFTKEY, V_FUNCTION, V_CCP and V_NUMPAD. Figure 5-2 shows the layout of the different keypad groups. This driver also maintains the state of the following keyboard modifier keys: <Ctrl>, left and right <Shift>, <Alt>, <Caps lock>, and <Num lock>. This state information is passed to the V_CCP, V_NUMPAD and V_QWERTY translator services.

Keyboard Translators

The keyboard translators act as subroutines for the logical keyboard driver. There are five translators corresponding to the keyboard keypads (see Figure 5-2). The five translators are:

V_QWERTY handles the Typewriter keypad.
V_FUNCTION handles the Compatibility Function keypad (F1 - F10).
V_NUMPAD handles the Numeric keypad (and its cursor keys).
V_SOFTKEY handles the HP Function keypad (f1 - f10)
V_CCP handles the Cursor Control keypad.

The translators for the HP Function keypad and Cursor Control keypad are special cases.

The V_SOFTKEY translator can translate its scancodes in the following ways:

1. Map function keys f1 thru f8 into function keys F1 thru F8 (V_SKEY2FKEY).

2. Throw away f1 thru f8 function keys (V_OFF).

3. Pass back f1 thru f8 function keys untranslated to the logical keyboard driver (V_RAW).

The V__CCP translator can translate its scancodes in the following ways:

1. Map Cursor Control keys to Numeric keypad cursor control scancodes (V__CCPCUR).

2. Map Cursor Control keys to Numeric keypad scancodes (V__CCPNUM).

3. Pass Cursor Control keys as untranslated scancodes to the logical keyboard driver (V__RAW).

4. Throw away all Cursor Control (CCP) keys (V__OFF).

Functions are provided by the STD-BIOS INT 16H driver to select any of the above mappings.

8042 Interface Driver

The 8042 interface driver (V__8042) sends translated scancodes to the 8042 controller chip. If the 8042 controller is busy this driver queues the scancode to be sent later when the 8042 controller is ready. In addition to passing scancodes from the keyboard to the 8042 controller, V__8042 processes keyboard controller commands to set keyboard LEDs and change keyboard typematic rates.

Data Structures

The EX-BIOS keyboard input system uses one data structure. The Keyboard ISR Event Record is a set of register definitions for inter-driver communication of input events. The following shows the Keyboard ISR Event Record definition.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard State (Only if state
               bit set in Data Type)

             Bit Data Definition

             07H  1   Left Unlabeled key pressed*
             06H  1   Right Unlabeled key pressed*
             05H  1   <Num lock> state active
             04H  1   <Caps lock> state active
             03H  1   <Ctrl> key pressed
             02H  1   Right <Shift> key pressed
             01H  1   Left <Shift> key pressed
             00H  1   <Alt> key pressed

          BL = Scancode

               Bit       Data   Definition

               07H        1     Break indicator
                          0     Make indicator
               06H-00H          Scancode
```

```
     CX = Number of bytes in buffer (scancode strings only)
     DH = Data Type
     DL = Logical keyboard drivers vector address / 6
     BP = HP-HIL device n vector address
  ES:SI = Pointer to buffer (scancode strings only)
```

* These keys are located to the immediate left and right of the space bar. They are only available on some international keyboards.

The Data Type field (DH) contains a code representing the current type of scancode contained in the ISR Event Record. When the logical keyboard driver calls a translator service, the Data Type will match the keypad group from which the scancode originated. After translation, the Data Type for the ISR Event Record returned to the logical keyboard driver should be T_KC_IBM_PC. See Table 5-13 for a complete list of keyboard event data types.

Table 5-13. Keyboard Event Data Types

Type	Definition	Value
T_KC_R0	Reserved	00H
T_KC_R1	Reserved	01H
T_KC_ASCII	ASCII data	02H
T_KC_R3	Reserved	03H
T_KC_ITF	HP150 keyboard (ITF) scancode	04H
T_KC_R5	Reserved	05H
T_KC_WILD	Device definable type	06H
T_KC_ENVOY	HP Vectra Keyboard set	07H
T_KC_IBM_AT	IBM-AT scancode set	08H
T_KC_BUTTON	Button data type	09H
T_KC_IBM_PC	IBM-PC scancode set	0AH
T_KC_HP_SOFTKEY	HP Function keypad (f1-f8)	0BH
T_KC_IS_FUNCTION	Compatibility Function keypad (F1-F10)	0CH
T_KC_HP_CCP	HP's Cursor Control keypad	0DH
T_KC_QWERTY	Typewriter keypad	0EH

Table 5-13. Keyboard Event Data Types (Cont.)

Type	Definition	Value
T_KC_NUMPAD	Numeric keypad	0FH
T_STRING	This is not a data type but an indicator bit for the keyboard data types only. If bit 4 is set, then the ISR Event record is for a string of scancodes pointed to by ES:SI and enumerated in CX; i.e., 00x1 ttttB indicates a string of data bytes of type defined by the lower nibble 'tttt'.	10H
T_STATE	This is not a data type but an indicator bit for the keyboard data types only. If bit 5 is set, it indicates that the corresponding ISR Event record contains the current state in BH.	20H

Logical Keyboard Driver

The logical keyboard driver determines the keypad group the scancode belongs to and sets the Data Type field in the ISR event record. Based on the Data Type a translator service is called to handle the scancode. For example, if the "Q" key scancode comes through, the logical keyboard driver determines the data type to be T_KC_QWERTY and calls the V_QWERTY translator. If the translator called by the logical keyboard driver is responsible for any of the keyboard modifier keys the current state variable is placed in the ISR Event Record and the state indicator bit is set in the Data Type field. Table 5-14 contains the scancode range to translator service assignments.

Table 5-14. Scancode Range to Translator Service Assignments

Driver Name	Scancode Range	Translation Performed
V_QWERTY	00H-36H	None
	38H-3AH	
	55H-5FH	
	6BH-6FH	
	78H-7FH	
V_SOFTKEY	70H-77H	3BH--42H (F1--F8)

Table 5-14. Scancode Range to Translator Service Assignments (Cont.)

Driver Name	Scancode Range	Translation Performed
V_FUNCTION	3BH-44H	None
V_NUMPAD	37H, 45H-54H	None
V_CCP	60H-6AH	Cursor Always - Regardless of state of the <Num lock> and <Shift> keys.

If the translation was successful, the returned ISR Event Record is passed to the logical keyboard drivers parent (V_8042).

Before passing a successful translation to its parent (V_8042) the logical keyboard driver performs two conditional tasks. First, it checks the state bit in the returned Data Type, if set the master copy of the keyboard state variable is updated with the copy returned in the ISR Event Record. Second, if the ISR event went to the V_CCP translator the logical keyboard driver takes the necessary steps to insure that cursor control keys are generated regardless of the <Num lock> and <Shift> key states.

If a translator wants to remove the scancode from the scancode stream it must return a status code of RS_DONE to the logical keyboard driver (See the CCP2GID driver in Appendix G).

Table 5-15 contains a summary of the logical keyboard driver functions.

Table 5-15. Logical Keyboard Driver Function Code Summary

Function Value	Function Equate	Definition
	F_Keyboard Driver	(This driver does not have a fixed HP_VECTOR_TABLE address)
00	F_ISR	Logical Interrupt
02	F_SYSTEM	System Intrinsics
02/00	SF_INIT	Driver initialization
02/06	SF_VERSION_DESC	Reports HP version number

Logical Keyboard Driver Function Definitions

F_ISR (AH = 00H)

This function processes the Keyboard ISR Event Record. It determines the range of the scancode, then calls the appropriate translation service.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard State (only if state bit set in Date type)
          BL = Scancode
          CX = Number of bytes in buffer (scancode strings only)
          DH = Scancode type
          DL = Vector address of keyboard / 6
          BP = HP-HIL device n vector address
       ES:SI = Pointer to buffer (scancode strings only)

On Exit: AH = Return Status Code

Registers Altered: AX, BX, CX, DX, SI,
                   BP, ES, DS
```

SF_INIT (AX = 0200H)

This subfunction is called to initialize the driver. Refer to Chapter 8 for a complete discussion of the protocol utilized in data space allocation ("last used DS" passed in register BX).

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BX = "Last used DS" in HP Data Area
          BP = HP-HIL device n vector address

On Exit: AH = Return Status Code
         BX = New "last used DS" is
              HP Data Area

Registers Altered: AX, BX, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = HP-HIL device n vector address

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current
              version number
      ES:DI = Pointer to the current version
              number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

Keyboard Translators

There is one keyboard translator service for each of the five keypad groups on the keyboard (see Figure 5-2). Two of the five services are special cases in that they are actually chains of translators to facilitate keyboard mapping. Figure 5-1 shows the translators and their mapping possibilities.

Applications may install routines to replace (or chain to) any one or all of the translators presented here. The INT 16H driver provides three functions to get the current HP_VECTOR_TABLE entries for the five keypad translators, to set these same values, and to reset them to their default values. The V_SYSTEM driver in Chapter 8 provides functions to get or set any fixed HP_VECTOR_TABLE entry (all EX-BIOS translators presented in this section have fixed entries). The V_SYSTEM functions allow replacement of translators other than the main five called by the logical keyboard driver (those in translator chains).

Applications that do not overlay existing translators, may install entirely new translators instead and map themselves into the HP Function and Cursor Control keypad translator chains as the parent drivers of the V_SOFTKEY and V_CCP services respectively. This method only works for the HP proprietary keypads.

V_SOFTKEY (BP = 003CH)

This translator service verifies the Data Type is T_KC_HP_SOFTKEY and then passes the ISR Event Record to its parent. By default, this translator is mapped to the V_SKEY2FKEY service; alternative mappings are presented in Table 5-16.

Table 5-16. V_SOFTKEY Driver Mapping Alternatives

Driver Name	Function
V_OFF	Discards the ISR event.
V_RAW	Returns the scancode untranslated.
V_SKEY2FKEY	Translates the HP Function keys into their respective Compatibility Function key equivalents.

F_ISR (AH = 00H)

This function verifies the passed in Data Type and passes the ISR event on to its parent.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard state (only if state bit set in type)
          BL = Scancode
          DH = Scancode type (T_KC_HP_SOFTKEY = 0BH)
          DL = Source vector address / 6
          BP = V_SOFTKEY (003CH)

On Exit: AH = Return Status Code
         BL = Translated scancode
         BH = New keyboard state (only if state bit set in type)
         DH = New scancode type (T_KC_IBM_PC = 0AH)

Registers Altered: AX, BX, DH, BP, DS
```

SF_INIT (AX = 0200H)

This subfunction is called to initialize the driver. Refer to Chapter 8 for a complete discussion of the protocol utilized in data space allocation ("last used DS" passed in register BX).

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BX = "Last used DS" in HP Data Area
          BP = V_SOFTKEY (003CH)

On Exit: AH = Return Status Code
         BX = "New last used DS" in HP Data Area

Registers Altered: AX, BX, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_SOFTKEY (003CH)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

V_QWERTY (BP = 0036H)

The V_QWERTY service verifies the correct Data Type. This service also maintains the state of the left and right <Shift> keys, the <Ctrl> key, the <Alt> key, the left and right unlabeled keys and the <Caps lock> key.

F_ISR (AH = 00H)

This function verifies the Data Type, updates the keyboard state variable, and returns.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard state (only if state bit set in type)
          BL = Scancode
          DH = Scancode type (T_KC_QWERTY = 0EH)
          DL = Source vector address / 6
          BP = V_QWERTY (0036H)

On Exit: AH = Return Status Code
         BH = New keyboard state (only if state bit set type)
         DH = New scancode type (T_KC_IBM_PC = 0AH)

Registers Altered: AX, BH, DH, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_QWERTY (0036H)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

V_FUNCTION (BP = 0042H)

This service verifies the Data Type, sets a new Data Type and returns.

F_ISR (AH = 00H)

This function verifies the Data Type, and sets the new one.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard state (only if state bit set in type)
          BL = Scancode
          DH = Scancode type (T_KC_IS_FUNCTION = 0CH)
          DL = Source vector address
          BP = V_FUNCTION (0042H)

On Exit: AH = Return status code
         DH = New scancode type (T_KC_IBM_PC = 0AH)

Registers Altered: AX, DH, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_FUNCTION (0042H)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

V_NUMPAD (BP = 0048H)

The V_NUMPAD service is the scancode translator for the numeric keypad. It verifies the Data Type is correct and maintains the state of the <Num lock> and <ScrLck> keys.

F_ISR (AH = 00H)

Verify Data Type and update state variable.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard state (only if state bit set in type)
          BL = Scancode
          DH = Scancode type (T_KC_NUMPAD = 0FH)
          DL = Source vector address / 6
          BP = V_NUMPAD (0048H)

On Exit: AH = Return status code
         BH = New keyboard state (only if state bit set in type)
         DH = New scancode type (T_KC_IBM_PC = 0AH)

Registers Altered: AX, BH, DH, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_NUMPAD (0048H)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

V_CCP (BP = 004EH)

This translator service verifies the Data Type is T_KC_HP_CCP and then passes the ISR Event Record to its parent. By default this translator is mapped to the V_CCPCUR service, alternative mappings are presented in Table 5-17.

Table 5-17. V_CCP Driver Mapping Alternatives

Driver Name	Function
V_OFF	Discards the ISR event.
V_RAW	Returns the scancode untranslated.
V_CCPNUM	Translates the cursor control pad scancodes into cursor or numeric key pad scancodes, depending on the <Num Lock> and <Shift> states.
V_CCPCUR	Translates the cursor control pad scancodes into cursor scancodes, regardless of the <Num Lock> and <Shift> states.

F_ISR (AH = 00H)

This function verifies the Data Type and passes the event to its parent.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard state (only if state bit set in type)
          BL = Scancode
          DH = Scancode type (T_KC_HP_CCP = 0DH)
          DL = Source vector address / 6
          BP = V_CCP (004EH)

On Exit:  AH = Return Status Code
          BL = Translated scancode
          BH = New keyboard state (only if state bit set in type)
          DH = New scancode type (T_KC_IBM_PC = 0AH)

Registers Altered: AX, BX, DH, BP, DS
```

SF_INIT (AX = 0200H)

This subfunction is called to initialize the driver. Refer to Chapter 8 for a complete discussion of the protocol utilized in data space allocation ("last used DS" passed in register BX).

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BX = "Last used DS" in HP Data Area
          BP = V_CCP (004EH)

On Exit: AH = Return Status Code
         BX = New "last used DS" in HP Data Area

Registers Altered: AX, BX, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_CCP (004EH)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

V_OFF Driver (BP = 0009CH)

The V_OFF driver effectively turns off any translator mapped to it. It returns a Return Status Code of RS_DONE, this indicates to the driver which called that all processing is complete, and to return. Returning this status code effectively terminates processing of the scancode.

F_ISR (AH = 00H)

This function sets a return status of RS_DONE and exits.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard state (only if state bit set in type)
          BL = Scancode
          DH = Scancode type (any type accepted)
          DL = Source vector address / 6
          BP = V_OFF (009CH)

On Exit: AH = RS_DONE

Registers Altered: AX, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_OFF (009CH)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

V_RAW Driver (BP = 0090H)

The V_RAW driver sets the data type to T_KC_IBM_PC (0AH) and returns, leaving the scancode untranslated.

F_ISR (AH = 00H)

This function sets a Data Type of T_KC_IBM_PC and a return status of RS_SUCCESSFUL.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard state (only if state bit set in type)
          BL = Scancode
          DH = Scancode type (any accepted)
          DL = Source vector address / 6
          BP = V_RAW (0090H)

On Exit: AH = Return Status Code
         DH = New scancode type (T_KC_IBM_PC = 0AH)

Registers Altered: AX, DH, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_RAW (0090H)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

V_CCPNUM (BP = 0096H)

The V_CCPNUM driver converts scancodes from the HP cursor control keypad to their respective Numeric keypad equivalents. The resultant scancodes will be either numeric or cursor scancodes, depending on the state of the <Num Lock> and <Shift> keys.

F_ISR (AH = 00H)

This function translates the scancode, sets a new Data Type and exits.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard state (only if state bit set in type)
          BL = Scancode
          DH = Scancode type (T_KC_HP_CCP = 0DH)
          DL = Source vector address / 6
          BP = V_CCPNUM (0096H)

On Exit: AH = Return Status Code
         BH = New keyboard state (only if state bit set in type)
         BL = Translated scancode
         DH = New scancode type (T_KC_IBM_PC = 0AH)

Registers Altered: AX, BX, DH, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_CCPNUM (0096H)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

V_CCPCUR (BP = 008AH)

The V_CCPCUR service converts scancodes from the Cursor Control Keypad to their respective numpad or cursor control equivalents. The <Shift> key states in the keyboard state variable are adjusted to cancel the effect of the <Num lock> key and force the Numeric keypad to operate in cursor mode. Upon return from this translator chain, the logical keyboard driver generates the appropriate <Shift> scancodes to account for the change to the keyboard state variable.

F_ISR (AH = 00H)

This function translates the scancode to its Numeric keypad equivalent, changes the Data Type to T_KC_IBM_PC, and adjusts the keyboard state variable to force the Numeric keypad into cursor mode.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard state (only if state bit set in type)
          BL = Scancode
          DH = Scancode type (T_KC_HP_CCP = 0DH)
          DL = Source vector address / 6
          BP = V_CCPCUR (008AH)

On Exit: AH = Return Status Code
         BH = New keyboard state (only if state bit set in type)
         BL = Translated scancode
         DH = New scancode type (T_KC_IBM_PC = 0AH)

Registers Altered: AX, BX, DH, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_CCPCUR (008AH)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

V_SKEY2FKEY (BP = 00A8H)

The V_SKEY2FKEY service translates HP Function key scancodes into their industry standard function key equivalents. The driver makes no attempt to verify that the scancode passed is in the range for an HP Function key.

F_ISR (AH = 00H)

This function translates the scancode, sets the Data Type to T_KC_IBM_PC and returns.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard state (only if state bit set in type)
          BL = Scancode
          DH = Scancode type (T_KC_HP_SOFTKEY = 0BH)
          DL = Source vector address / 6
          BP = V_SKEY2FKEY (00A8H)

On Exit: AH = Return Status Code
         BL = Translated scancode
         DH = New scancode type (T_KC_IBM_PC = 0AH)

Registers Altered: AX, BL, DH, BP, DS
```

SF_VERSION_DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_SKEY2FKEY (00A8H)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

V_8042 Driver (BP = 00AEH)

This driver provides an interface to the 8042 keyboard controller chip. It responds to 8042 service requests and Input System logical interrupt requests (F_ISR's) to output scancodes to the 8042 chip. It also provides an application interface to 8042 timer services and switch settings. Table 5-18 contains a function code summary for this driver.

Table 5-18. V_8042 Driver Function Code Summary

Func. Value	Function Equate	Definition
	V_8042	8042/keyboard interface. provides HP extensions to INT 16H
00	F_ISR	Processes ISR event record
02	F_SYSTEM	System functions
02/00	SF_INIT	Initialize driver
02/02	SF_START	Driver start-up
02/06	SF_VERSION _DESC	Report HP version number
04	F_IO_CONTROL	Driver dependent functions
04/00-0		Reserved
04/0A	SF_CREAT _INTR	Create interval entry
04/0C	SF_DELET _INTR	Delete interval entry
04/0E	SF_ENABL _INTR	Enable interval
04/10	SF_DISBL _INTR	Disable interval
04/12	SF_SET _RAMSW	Set RAM switch to one (1)
04/14	SF_CLR _RAMSW	Set RAM switch to zero (0)
04/16	SF_SET _CRTSW	Set CRT switch to one (1)
04/18	SF_CLR _CRTSW	Set CRT switch to zero (0)
04/1A	SF_PASS _THRU	Pass data byte to 8042

V_8042 Driver Function Definitions

F_ISR (AH = 00H)

This function processes a Keyboard ISR Event Record. It checks to see if the 8042 will accept another scancode. If not, the scancode is placed in a queue. If the 8042 can accept a scancode, it writes the scancode out. The scancode queue has room for 127 entries plus one overrun character.

```
On Entry: AH = F_ISR (00H)
          BH = Keyboard state (only if state bit set in type)
          BL = Scancode
          CX = Number of scancodes in buffer (string type only)
          DH = Scancode type
          DL = Source vector address / 6
          BP = V_8042 (00AEH)
       ES:SI = Pointer to buffer (string type only)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_INIT (AX = 0200H)

This subfunction is called to initialize the driver. Refer to Chapter 8 for a complete discussion of the protocol utilized in data space allocation ("last used DS" passed in register BX).

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BX = "Last used DS" in HP Data Area
          BP = V_8042 (00AEH)

On Exit: AH = Return Status Code
         BX = New "last used DS" in HP Data Area

Registers Altered: AX, BX, BP, DS
```

SF_START (AX = 0202H)

This subfunction starts the 8042 driver.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_START (02H)
          BP = V_8042 (00AEH)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF__VERSION__DESC (AX = 0206H)

This subfunction returns the release date code and a double word pointer to the current version number. The date code consists of two BCD coded bytes containing the year and week of release. The BL register contains the number of years since 1960 and the BH register contains the week of the year.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = V_8042 (00AEH)

On Exit: AH = Return Status Code
         BX = Release date code
         CX = Number of bytes in current version number
      ES:DI = Pointer to the current version number

Registers Altered: AX, BX, CX, DI, ES, BP, DS
```

SF__CREAT__INTR (AX = 040AH)

The 8042 driver will call up to eight drivers at 1/60 second intervals. This subfunction creates an entry in the table of driver vectors which are called. Note that this subfunction only creates the entry; it does not enable the interval service. This is accomplished with the SF__ENABL__INTR subfunction.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CREAT_INTR (0AH)
          BH = Vector number (vector address divided by six of
               driver requesting service
          BP = V_8042 (00AEH)

On Exit: AH = Return Status Code
              RS_FAIL indicates driver vector table full.

Registers Altered: AX, BP, DS
```

SF__DELET__INTR (AX = 040CH)

This function removes the passed in vector number from the interval service table.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_DELET_INTR (0CH)
          BH = Vector number (vector address divided by six) of
               driver to delete from table
          BP = V_8042 (00AEH)

On Exit: AH = Return Status Code
              RS_FAIL indicates vector not in table.

Registers Altered: AX, BP, DS
```

SF_ENABL_INTR (AX = 040EH)

This function enables interrupt service for a driver. The vector number passed is checked against the table. If an entry with that vector number is found, interval service is enabled. When the interval expires all enabled drivers in the list will be interrupted with a function code of F_SYSTEM (02H) in AH and a subfunction code of SF_INTERVAL (14H) in AL.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_ENABL_INTR (0EH)
          BH = Vector number (vector address divided by six) of
               driver requesting service
          BP = V_8042 (00AEH)

On Exit: AH = Return Status Code
              RS_FAIL indicates vector not in table.

Registers Altered: AX, BP, DS
```

SF_DISBL_INTR (AX = 0410H)

This function disables interrupt service for a driver. The vector number passed is checked against the table. If an entry with that vector number is found, interval service is disabled.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_DISBL_INTR (10H)
          BH = Vector number (vector address divided by six) of
               driver to be disabled
          BP = V_8042 (00AEH)

On Exit: AH = Return Status Code
              RS_FAIL indicates vector not in table.

Registers Altered: AX, BP, DS
```

SF_SET_RAMSW (AX = 0412H)

This function sets the industry standard extended RAM "switch" in the 8042 status register. This switch indicates that the second 256K RAM bank on the system board is enabled (default condition).

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_SET_RAMSW (12H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_CLR_RAMSW (AX = 0414H)

This function clears the industry standard extended RAM "switch" in the 8042 status register. When this switch is off it indicates that the second 256K RAM bank is disabled.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CLR_RAMSW (14H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_SET_CRTSW (AX = 0416H)

This function sets the industry standard primary CRT "switch" in the 8042 status register. When the switch is set it indicates the primary display is attached to the Multimode graphics adapter (Default condition).

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_SET_CRTSW (16H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_CLR_CRTSW (AX = 0418H)

This function clears the industry standard primary CRT "switch" in the 8042 status register. When this switch is clear it indicates the primary display is attached to the monochrome display adapter.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_CLR_CRTSW (18H)

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

SF_PASS_THRU (AX = 041AH)

This function outputs the byte in BL to the 8042 using the pass thru command to prevent the 8042 from interpreting the data as a scancode or a command.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_PASS_THRU (1AH)
          BL = data byte to pass thru the 8042

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS
```

8042 Keyboard Controller

This section discusses the role of the 8042 keyboard controller. The information presented applies to both the **Vectra Enhanced keyboard** and **Vectra Keyboard/DIN**, unless indicated otherwise.

Overview

The primary function of the 8042 keyboard controller is to manage the industry standard keyboard interface. (Directly accessing this hardware interface may affect program portability and is **not recommended**.) The 8042 keyboard controller also acts as a loopback buffer for the input system to the STD-BIOS keyboard driver. The 8042 is implemented in such a way as to maintain standard IBM PC/AT compatibility, while at the same time supporting all of the features of the input system.

The 8042 keyboard controller accepts two sets of industry standard commands from the STD-BIOS drivers that control the operation of the controller and the keyboard itself. One set is controller commands, the other is keyboard commands (both sets are listed in Table 5-19). Controller commands are executed by the 8042 controller while keyboard commands are sent to the keyboard for execution.

8042 Controller and Keyboard Commands

Each of the controller command and keyboard command sets has its own protocol. The 8042 has two ports: a command port (I/O address 64H), and a data port (I/O address 60H). 8042 controller commands are written to the command port. If the command has parameters associated with it, the parameters are written to the data port. Keyboard commands are written to the data port. If the command has parameters associated with it, they are also written to the data port. All data written to the data port is interpreted as a keyboard command unless the previous command written to the command or data port required parameters.

The following code writes a one-byte command to the 8042 controller to disable the keyboard interface.

```
hp8042_cmd_port       equu    64h        ; IBM cmd/status port
hp8042_status_port    equ     64h        ; IBM cmd/status port
hp8042_data_port      equ     60h        ; IBM data port
hp8042_ibf_mask       equ     02h        ; Input buffer full mask

hp8042_iface_dis      equ     0ADh       ; Disable interface

dis_8042     proc     near
             push     cx                 ; save working set of regs
             push     ax
             xor      cx,cx              ; loop 64k times (if necessary)
             cli                         ; ints must be off for this loop
```

```
dis_8042_10:
            in    al,hp8042_status_port     ; get status and see if 80286
            test al,hp8042_ibf_mask         ; input buffer if full
            loopnz dis_8042_10              ; loop if it is

            mov     al,hp8042_iface_dis     ; load disable command and
            out     hp8042_cmd_port,al      ; ship it out
            sti

            pop     ax
            pop     cx
            ret
dis_8042    endp
```

The following code writes a two byte command to the 8042 to turn on all the keyboard LEDs at once.

```
hp8042_cmd_port     equ   64h               ; Hp8042 cmd/status port
hp8042_status_port  equ   64h               ; Hp8042 cmd/status port
hp8042_data_port    equ   60h               ; Hp8042 data port
hp8042_set_led      equ   0edh              ; Set keyboard leds command

hp8042_ibf_mask     equ   02h               ; Input buffer full mask
led_data            equ   07h               ; Led mask to send out

set_8042    proc    near
            push    cx                      ; save working set of regs
            push    bx
            push    ax
            xor     cx,cx                   ; loop 64k times (if necessary)
            mov     bh,led_data             ; load data for loop
            mov     bl,hp8042_set_led       ; load command
            cli                             ; ints must be off for this loop


set_8042_10:
            in      al,hp8042_status_port ; get status and see if 8042
            test    al,hp8042_ibf_mask    ; input buffer if full
            loopnz set_8042_10            ; loop if it is

            mov     al,bl                   ; load command and
            out     hp8042_data_port,al     ; ship it out
            cmp     bh,al                   ; did we output both bytes
            je      set_8042_20             ; yes, skip out
            mov     bl,bh                   ; set up for next iteration
            xor     cx,cx
            jmp     short set_8042_10       ; loop
```

```
set_8042_20:
            sti                                ; CHANGE this to restore
                                               ; int flag to previous state
                                               ; instead of on (if needed)
            pop     ax
            pop     bx
            pop     cx
            ret
set_8042    endp
```

Table 5-19 lists the 8042 controller commands. These commands are categorized as READ, SNGL, or DBL. READ commands cause the 8042 controller to place the indicated data byte in it's output buffer, input port 60H, to be read by the CPU. SNGL commands are commands written to output port 64H. DBL byte commands are written to output port 64H with the following data byte being written to output port 60H.

Table 5-19. 8042 Controller Commands

Command	Type	Description
020H	READ	Reads byte zero of the 8042's internal RAM. This byte is the last keyboard command sent to the 8042.
021H-03FH	READ	Reads the byte specified by the lower five bits of the command in the 8042's internal RAM. E.g. 8042 controller command 34H will report contents of the 14H byte of the 8042's RAM.
060H-07FH	DBL	Writes the data byte to the address specified in the low five bits of the command.
0AAH	SNGL	Initiate Self-Test. This command instructs the 8042 to perform a self test. If no errors are detected, 55H is returned in the data port.
0ABH	SNGL	Initiate Interface Test. This command instructs the 8042 to test the interface between itself and the keyboard. (Always returns 0 = successful)
0ACH	READ	Diagnostic Dump. The contents of the 8042 internal RAM registers (16 bytes), output port, input port, and status word are sent to the system. All diagnostic data is sent to the system in the same manner as scancodes. (Not supported)
0ADH	SNGL	Disable Keyboard. This command disables the keyboard. Bit 4 of the current command byte will be set to '1' in the 8042. This is equivalent to issuing a command byte with bit 4 set to '1'. Note that this command will have no effect if bit 3 of the command byte is set to '1'.

Table 5-19. 8042 Controller Commands (Cont.)

Command	Type	Description
0AEH	SNGL	Enable Keyboard. This command re-enables the keyboard. Bit 4 of the current command byte is cleared in the 8042 This is equivalent to issuing a command byte with bit 4 set to '0'.
0C0H	READ	Read Input Port. The current value of the input port is returned. Bit 7 indicates the status of the front panel keylock. Bits 0 - 3 will always be reported as '1'. Bits 4 - 6 are undefined.
0D0H	READ	Read Output Port. The current value of the output port is returned. See Table 5-21 for bit definitions.
0D1H	DBL	Write Output Port. The next byte written to the data port will be written to the 8042 output port. The bit definitions for this port are given in Table 5-21. WARNING - The System Reset bit should not be written low. To reset the system, use the Pulse Output Port command.
0DDH	SNGL	Disable Address Bit 20. Disables the A20 address of the processor address bit. This is the normal state of this pin the in real addressing mode.
0DFH	SNGL	Enable address Bit 20. Enables the A20 address of the processor address bit. This state is only used in protected mode.
0E0H	READ	Read Test Inputs. This command will output the current state of the 8042 test inputs, T0 and T1. The current state of T0 is stored in bit 0 and T1 in bit 1. Both bits will be reported as '1', unless the keyboard interface is inhibited. Bits 2 through 7 are undefined.
0F0H-0FFH	SNGL	Pulse Output Port. Bits 0 - 3 of the output port may be pulsed low for approximately 6 microseconds. Bits 0 through 3 contain a mask which is interpreted by the 8042 to determine which bits are pulsed. A bit is pulsed if its corresponding mask bit is '0'; if it is '1' its current state is maintained. Note - The System Reset bit is connected to bit 0. If the system needs to be reset, this command should be used (i.e., the bit should be pulsed, not brought low indefinitely.)

Table 5-20 indicates the format of the data byte written to the 8042 Controller subsequent to the 8042 Command 20H listed in Table 5-19.

Table 5-20. 8042 Command Byte Format

Bit	Data	Definition
07H	0	Reserved--must always be 0.
06H		Scancode conversion mode.
	1	The scancodes received from the keyboard are converted into PC/XT scancodes.
	0	Convert to AT scancodes.
05H		Acts as a NOP (No Operation instruction).
04H		Disable Keyboard. Data will not be sent or received by the keyboard.
	1	Disables the keyboard.
	0	Restore operation.
03H		Inhibit override.
	1	Prevents the keyboard from being disabled via the computer's Security Keylock.
02H		System Flag. The value of this bit is stored as the System Flag Bit. This bit may be read via port 60H.
01H		Reserved--must always be 0.
	1	Instructs the 8042 to issue an OBF (Output Buffer Full) interrupt when data is in the output buffer.
	0	Disables this feature.

Table 5-21 indicates the format of the data byte written to the 8042 controller subsequent to the 8042 Command Write Output Port 0D1H, or read from the 8042 controller subsequent to the 8042 Command Read Output Port 0D0H.

Table 5-21. 8042 Command Output Port Bit Mask

Bit	Data	Definition
07H	1	Keyboard data line
06H	1	Keyboard clock line
05H	1	Undefined
04H	1	Output Buffer Full Interrupt (OBF)
03H	1	Undefined
02H	1	Undefined
01H	1	A20 Gate
00H	1	System Reset

Table 5-22 lists the keyboard commands. These commands are categorized as SNGL or DBL. SNGL commands are commands written to output port 60H. DBL byte commands are written to output port 60H with the subsequent data byte, also, being written to output port 60H. The coding examples given for 8042 controller commands is similar to the procedure for writing keyboard commands. The notable exception being the I/O address 60H is substituted for the I/O address 64H (defined with the equate, hp8042_cmd_port).

Table 5-22. Keyboard Commands

Command	Type	Description
0EDH	DBL	Set/Reset Mode Indicators. The keyboard has three status indicators; Caps lock, Num lock, and Scroll lock. This command is used to turn these indicators on and off. After the command is issued, the system must wait for an ACK (0FAH in Table 5-28) from the keyboard (see below). When it is received, a second byte is issued to the keyboard. Bits 0 - 2 represent Scroll lock, Num lock, and Caps lock, respectively. Setting their respective bits to 1 turns the indicator on, while a 0 turns it off. Bits 3 - 7 should be set to 0. (See Table 5-23)
0EEH	SNGL	Echo. This is a diagnostic tool. When this command is issued, the keyboard returns an EEH.
0EFH	SNGL	No Operation (NOP). These codes are reserved for future use. The keyboard will acknowledge these codes, but no other action will be performed.
0F0H	DBL	Select Alternate Scancodes. This command instructs the keyboard to select one of three sets of scancodes. When the keyboard receives this command it responds with an 'ACK' and clears the output buffer and the typematic key (if one is active). The system then sends the option byte to select the appropriate scancode set: 01H selects set 1, 02H selects set 2, 03H selects set 3. The keyboard responds to this with another 'ACK'. (See Tables 5-24a, 5-24b, and 5-24c).
0F1H	SNGL	No Operation (NOP). These codes are reserved for future use. The keyboard will acknowledge these codes, but no other action will be performed.
0F2H	SNGL	Request Keyboard Identification information. The keyboard responds with an 'ACK', discontinues scanning and sends the two keyboard ID bytes. The second byte must follow the first by no more then 500 microseconds. After the output of the second ID byte, the keyboard resumes scanning.
0F3H	DBL	Set Typematic Rate/Delay. This command sets the values for the typematic rate and delay. The typematic rate is the number of make scancodes per second sent in the typematic (repeat) mode. The delay is the amount of time a key must be held down until it enters the typematic mode.

Table 5-22. Keyboard Commands (Cont.)

Command	Type	Description
		The rate and delay are passed in the next byte after the command. Bits 0 through 4 contain the rate and bits 5 and 6 contain the delay. Bit 7 is unused. The 8042 chip accepts STD AT typematic commands which are composed of two bits of delay (6,5) and five bits of rate (4 - 0). The two low order bits of the rate value are stripped off by the 8042 and the result translated into the typematic rate. (See Table 5-27.)
0F4H	SNGL	Enable. This command enables keyboard action. The keyboard will issue an 'ACK' response, then begin sending scancodes as keys are pressed.
0F5H	SNGL	Default Disable. This command sets the keyboard parameters to their power-on default state and disables the transmission of scancodes. The keyboard will send an 'ACK' response to this command.
0F6H	SNGL	Set Default. This command sets the keyboard parameters to their power-on state and sends an 'ACK' response. the keyboard will continue to transmit scancodes after receipt of this command.
0F7H	SNGL	Set All Keys Typematic. When the keyboard receives this command it responds with an 'ACK', clears output buffers, sets all keys to typematic and continues scanning. This command can be sent using any scancode set, but only set 3 is affected.
0F8H	SNGL	Set All Keys Make/Break. When the keyboard receives this command it responds with an 'ACK', clears output buffers, sets all keys to make/break and continues scanning. This command can be sent using any scancode set, but only set 3 is affected.
0F9H	SNGL	Set All Keys Make. When the keyboard receives this command it responds with an 'ACK', clears output buffers, sets all keys to make and continues scanning. This command can be sent using any scancode set, but only set 3 is affected.
0FAH	SNGL	Set All Keys Typematic/Make/Break. When the keyboard receives this command it responds with an 'ACK', clears output buffers, sets all keys to typematic/make/break and continues scanning. This command can be sent using any scancode set, but only set 3 is affected.
0FBH	DBL	Set Key Typematic. When the keyboard receives this command it clears output buffers to receive key ID. The system identifies each key by its scancode set 3 value (the only valid means of key identification). Each identified key is set to typematic.

Table 5-22. Keyboard Commands (Cont.)

Command	Type	Description
0FCH	DBL	Set Key Make/Break. When the keyboard receives this command it clears output buffers to receive key ID. The system identifies each key by its scancode set 3 value (the only valid means of key identification). Each identified key is set to make/break.
0FDH	DBL	Set Key Make. When the keyboard receives this command it clears output buffers to receive key ID. The system identifies each key by its scancode set 3 value (the only valid means of key identification). Each identified key is set to make.
0FEH	SNGL	Resend. This command may be sent to the keyboard whenever an error is detected by the system. This command must be sent before the next scancode is to be transmitted. If the last code sent by the keyboard was a Resend command, the keyboard will send the prior code.
0FFH	SNGL	Reset. This command instructs the keyboard to perform its Power-On Reset function. This step takes at least 300 milliseconds, during which the keyboard is disabled.

Table 5-23 indicates the format of the data byte written to the output port 60H subsequent to the Keyboard Command 'Set Mode Indicators' 0EDH.

Table 5-23. Set Mode Indicators Data Byte Format

Bit	Data	Definition
07H-03H		Reserved, should be set to zero
02H		Caps Lock Mode Indicator
	0	Turns off Caps Lock indicator
	1	Turns on Caps Lock Indicator
01H		Num Lock Mode Indicator
	0	Turn off Num Lock indicator
	1	Turn on Num Lock indicator
00H		Scroll Lock Mode Indicator
	0	Turn off Scroll Lock indicator
	1	Turn on Scroll Lock indicator

Tables 5-24, 5-25 and 5-26 list the three scancode sets that can be switched to and from by the keyboard command 0F0H (select alternate scancodes). The system defaults to scancode set 2.

Scancode Set 1

In this set, keys are assigned a base scancode (extra codes generate artificial shift states in the system, in some cases). The typematic scancodes are identical to the base scancode for each key.

In part one of the following table, keys send the codes shown (regardless of any shift states). Refer to Figures 5-2 and 5-3 for keyboard layouts showing the associated key numbers.

TABLE 5-24a. SCANCODE SET 1 (PART 1)

Key Number	Make Code	Break Code	Key Number	Make Code	Break Code
1	29	A9	49	2F	AF
2	02	82	50	30	B0
3	03	83	51	31	B1
4	04	84	52	32	B2
5	05	85	53	33	B3
6	06	86	54	34	B4
7	07	87	55	35	B5
8	08	88	57	36	B6
9	09	89	58	1D	9D
10	0A	8A	59 ***	5E	DE
11	0B	8B	60	38	B8
12	0C	8C	61	39	B9
13	0D	8D	62	E0 38	E0 B8
15	0E	8E	63 ***	5F	DF
16	0F	8F	64	E0 1D	E0 9D
17	10	90	90	45	C5
18	11	91	91	47	C7
19	12	92	92	4B	CB
20	13	93	93	4F	CF
21	14	94	96	48	C8
22	15	95	97	4C	CC
23	16	96	98	50	D0
24	17	97	99	52	D2
25	18	98	100	37	B7
26	19	99	101	49	C9
27	1A	9A	102	4D	CD
28	1B	9B	103	51	D1
29 *	2B	AB	104	53	D3
30	3A	BA	105	4A	CA
31	1E	9E	106	4E	CE
32	1F	9F	108	E0 1C	E0 9C
33	20	A0	110	01	81
34	21	A1	112	3B	BB
35	22	A2	113	3C	BC
36	23	A3	114	3D	BD
37	24	A4	115	3E	BE
38	25	A5	116	3F	BF
39	26	A6	117	40	C0
40	27	A7	118	41	C1
41	28	A8	119	42	C2
42 **	2B	AB	120	43	C3
43	1C	9C	121	44	C4
44	2A	AA	122	57	D7
45 **	56	D6	123	58	D8
46	2C	AC	125	46	C6

* 101-key keyboard only.
** 102-key keyboard only (non-US).
*** Asian keyboard only.

The next parts show a series of codes dependent on the state of the keys <Ctrl>, <Alt>, <Shift> and <Num Lock>. Since the base scancode is the same as that of another key, an extra code (E0 hex) has been added to the base to make it unique.

TABLE 5-24b. SCANCODE SET 1 (PART 2)

Key Number	Base Case, or Shift + Num Lock Make / Break	Shift Case Make / Break *	Num Lock on Make / Break
75	E0 52 / E0 D2	E0 AA E0 52 / E0 D2 E0 2A	E0 2A E0 52 / E0 D2 E0 AA
76	E0 53 / E0 D3	E0 AA E0 53 / E0 D3 E0 2A	E0 2A E0 53 / E0 D3 E0 AA
79	E0 4B / E0 CB	E0 AA E0 4B / E0 CB E0 2A	E0 2A E0 4B / E0 CB E0 AA
80	E0 47 / E0 C7	E0 AA E0 47 / E0 C7 E0 2A	E0 2A E0 47 / E0 C7 E0 AA
81	E0 4F / E0 CF	E0 AA E0 4F / E0 CF E0 2A	E0 2A E0 4F / E0 CF E0 AA
83	E0 48 / E0 C8	E0 AA E0 48 / E0 C8 E0 2A	E0 2A E0 48 / E0 C8 E0 AA
84	E0 50 / E0 D0	E0 AA E0 50 / E0 D0 E0 2A	E0 2A E0 50 / E0 D0 E0 AA
85	E0 49 / E0 C9	E0 AA E0 49 / E0 C9 E0 2A	E0 2A E0 49 / E0 C9 E0 AA
86	E0 51 / E0 D1	E0 AA E0 51 / E0 D1 E0 2A	E0 2A E0 51 / E0 D1 E0 AA
89	E0 4D / E0 CD	E0 AA E0 4D / E0 CD E0 2A	E0 2A E0 4D / E0 CD E0 AA

* The AA/2A shift make and break is sent with the other scancodes if the left Shift key is held down. If the right Shift key is held down, then B6/36 is sent. Both sets of codes are sent with the other scancode if both Shift keys are held down.

TABLE 5-24c. SCANCODE SET 1 (PART 3)

Key Number	Scancode Make / Break	Shift Case Make / Break *
95	E0 35 / E0 B5	E0 AA E0 35 / E0 B5 E0 2A

* The AA/2A shift make and break is sent with the other scancodes if the left Shift key is held down. If the right Shift key is held down, then B6/36 is sent. Both sets of codes are sent with the other scancode if both Shift keys are held down.

TABLE 5-24d. SCANCODE SET 1 (PART 4)

Key Number	Scancode Make / Break	Ctrl Case, Shift Case Make / Break	Alt Case Make / Break
124	E0 2A E0 37 / E0 B7 E0 AA	E0 37 / E0 B7	54 / D4

TABLE 5-24e. SCANCODE SET 1 (PART 5)

Key Number	Make Code	Ctrl Key Pressed
126 *	E1 1D 45 E1 9D C5	E0 46 E0 C6

* Not a Typematic key. All associated scancodes occur on the make of the key.

Scancode Set 2

In this set, when a key is pressed, each key is assigned a unique 8-bit make scancode. Each key also sends a break code when the key is released. The break code is made up of 2 bytes: the first being the break code prefix (F0 hex), and the second being the make scancode for that key.

In part one of the following table, keys send the codes shown (regardless of any shift states). Refer to Figures 5-2 and 5-3 for keyboard layouts showing the associated key numbers.

TABLE 5-25a. SCANCODE SET 2 (PART 1)

Key Number	Make Code	Break Code	Key Number	Make Code	Break Code
1	0E	F0 0E	48	21	F0 21
2	16	F0 16	49	2A	F0 2A
3	1E	F0 1E	50	32	F0 32
4	26	F0 26	51	31	F0 31
5	25	F0 25	52	3A	F0 3A
6	2E	F0 2E	53	41	F0 41
7	36	F0 36	54	49	F0 49
8	3D	F0 3D	55	4A	F0 4A
9	3E	F0 3E	57	59	F0 59
10	46	F0 46	58	14	F0 14
11	45	F0 45	59 ***	5E	F0 5E
12	4E	F0 4E	60	11	F0 11
13	55	F0 55	61	29	F0 29
15	66	F0 66	62	E0 11	E0 F0 11
16	0D	F0 0D	63 ***	5F	F0 5F
17	15	F0 15	64	E0 14	E0 F0 14
18	1D	F0 1D	90	77	F0 77
19	24	F0 24	91	6C	F0 6C
20	2D	F0 2D	92	6B	F0 6B
21	2C	F0 2C	93	69	F0 69
22	35	F0 35	96	75	F0 75
23	3C	F0 3C	97	73	F0 73
24	43	F0 43	98	72	F0 72
25	44	F0 44	99	70	F0 70
26	4D	F0 4D	100	7C	F0 7C
27	54	F0 54	101	7D	F0 7D
28	5B	F0 5B	102	74	F0 74
29 *	5D	F0 5D	103	7A	F0 7A
30	58	F0 58	104	71	F0 71
31	1C	F0 1C	105	7B	F0 7B
32	1B	F0 1B	106	79	F0 79
33	23	F0 23	108	E0 5A	E0 F0 5A
34	2B	F0 2B	110	76	F0 76
35	34	F0 34	112	05	F0 05
36	33	F0 33	113	06	F0 06
37	3B	F0 3B	114	04	F0 04
38	42	F0 42	115	0C	F0 0C
39	4B	F0 4B	116	03	F0 03
40	4C	F04C	117	0B	F0 0B
41	52	F0 52	118	83	F0 83
42 **	5D	F0 5D	119	0A	F0 0A
43	5A	F05A	120	01	F0 01
44	12	F0 12	121	09	F0 09
45 **	61	F0 61	122	78	F0 78
46	1A	F0 1A	123	07	F0 07
47	22	F0 22	125	7E	F0 7E

* 101-key keyboard only.
** 102-key keyboard only (non-US).
*** Asian keyboard only.

The next parts show a series of codes dependent on the state of the keys <Ctrl>, <Alt>, <Shift> and <Num Lock>. Since the base scancode is the same as that of another key, an extra code (E0 hex) has been added to the base to make it unique.

TABLE 5-25b. SCANCODE SET 2 (PART 2)

Key Number	Base Case, or Shift + Num Lock Make / Break	Shift Case Make / Break *	Num Lock on Make / Break
75	E0 70 / E0 F0 70	E0 F0 12 E0 70 / E0 F0 70 E0 12	E0 12 E0 70 / E0 F0 70 E0 F0 12
76	E0 71 / E0 F0 71	E0 F0 12 E0 71 / E0 F0 71 E0 12	E0 12 E0 71 / E0 F0 71 E0 F0 12
79	E0 6B / E0 F0 6B	E0 F0 12 E0 6B / E0 F0 6B E0 12	E0 12 E0 6B / E0 F0 6B E0 F0 12
80	E0 6C / E0 F0 6C	E0 F0 12 E0 6C / E0 F0 6C E0 12	E0 12 E0 6C / E0 F0 6C E0 F0 12
81	E0 69 / E0 F0 69	E0 F0 12 E0 69 / E0 F0 69 E0 12	E0 12 E0 69 / E0 F0 69 E0 F0 12
83	E0 75 / E0 F0 75	E0 F0 12 E0 75 / E0 F0 75 E0 12	E0 12 E0 75 / E0 F0 75 E0 F0 12
84	E0 72 / E0 F0 72	E0 F0 12 E0 72 / E0 F0 72 E0 12	E0 12 E0 72 / E0 F0 72 E0 F0 12
85	E0 7D / E0 F0 7D	E0 F0 12 E0 7D / E0 F0 7D E0 12	E0 12 E0 7D / E0 F0 7D E0 F0 12
86	E0 7A / E0 F0 7A	E0 F0 12 E0 7A / E0 F0 7A E0 12	E0 12 E0 7A / E0 F0 7A E0 F0 12
89	E0 74 / E0 F0 74	E0 F0 12 E0 74 / E0 F0 74 E0 12	E0 12 E0 74 / E0 F0 74 E0 F0 12

* The F0 12/12 shift make and break is sent with the other scancodes if the left Shift key is held down. If the right Shift key is held down, then F0 59/59 is sent. Both sets of codes are sent with the other scancode if both Shift keys are held down.

TABLE 5-25c. SCANCODE SET 2 (PART 3)

Key Number	Scancode Make / Break	Shift Case Make / Break *
95	E0 4A / E0 F0 4A	E0 F0 12 4A / E0 12 F0 4A

* The F0 12/12 shift make and break is sent with the other scancodes if the left Shift key is held down. If the right Shift key is held down, then F0 59/59 is sent. Both sets of codes are sent with the other scancode if both Shift keys are held down.

TABLE 5-25d. SCANCODE SET 2 (PART 4)

Key Number	Scancode Make / Break	Ctrl Case, Shift Case Make / Break	Alt Case Make / Break
124	E0 12 E0 7C / E0 F0 7C E0 F0 12	E0 7C / E0 F0 7C	84 / F0 84

TABLE 5-25e. SCANCODE SET 2 (PART 5)

Key Number	Make Code	Ctrl Key Pressed
126 *	E1 14 77 E1 F0 14 F0 77	E0 7E E0 F0 7E

* Not a Typematic key. All associated scancodes occur on the make of the key.

Scancode Set 3

In this set, when a key is pressed, each key is assigned a unique 8-bit make scancode. Each key also sends a break code when the key is released. The break code is made up of 2 bytes: the first being the break code prefix (F0 hex), and the second being the make scancode for that key.

In part one of the following table, keys send the codes shown (regardless of any shift states). Refer to Figures 5-2 and 5-3 for keyboard layouts showing the associated key numbers.

TABLE 5-26. SCANCODE TABLE: SET 3

Key Number	Make Code	Break Code	Default Key State	Key Number	Make Code	Break Code	Default Key State
1	0E	F0 0E	Typematic	55	4A	F0 4A	Typematic
2	16	F0 16	Typematic	57	59	F0 59	Make/Break
3	1E	F0 1E	Typematic	58	11	F0 11	Make/Break
4	26	F0 26	Typematic	59 ***	18	F0 18	Typematic
5	25	F0 25	Typematic	60	19	F0 19	Make/Break
6	2E	F0 2E	Typematic	61	29	F0 29	Typematic
7	36	F0 36	Typematic	62	39	F0 39	Make only
8	3D	F0 3D	Typematic	63 ***	38	F0 38	Typematic
9	3E	F0 3E	Typematic	64	58	F0 58	Make only
10	46	F0 46	Typematic	75	67	F0 67	Make only
11	45	F0 45	Typematic	76	64	F0 64	Typematic
12	4E	F0 4E	Typematic	79	61	F0 61	Typematic
13	55	F0 55	Typematic	80	6E	F0 6E	Make only
15	66	F0 66	Typematic	81	65	F0 65	Make only
16	0D	F0 0D	Typematic	83	63	F0 63	Typematic
17	15	F0 15	Typematic	84	60	F0 60	Typematic
18	1D	F0 1D	Typematic	85	6F	F0 6F	Make only
19	24	F0 24	Typematic	86	6D	F0 6D	Make only
20	2D	F0 2D	Typematic	89	6A	F0 6A	Typematic
21	2C	F0 2C	Typematic	90	76	F0 76	Make only
22	35	F0 35	Typematic	91	6C	F0 6C	Make only
23	3C	F0 3C	Typematic	92	6B	F0 6B	Make only
24	43	F0 43	Typematic	93	69	F0 69	Make only
25	44	F0 44	Typematic	95	77	F0 77	Make only
26	4D	F0 4D	Typematic	96	75	F0 75	Make only
27	54	F0 54	Typematic	97	73	F0 73	Make only
28	5B	F0 5B	Typematic	98	72	F0 72	Make only
29 *	5C	F0 5C	Typematic	99	70	F0 70	Make only
30	14	F0 14	Make/Break	100	7E	F0 7E	Make only
31	1C	F0 1C	Typematic	101	7D	F0 7D	Make only
32	1B	F0 1B	Typematic	102	74	F0 74	Make only
33	23	F0 23	Typematic	103	7A	F0 7A	Make only
34	2B	F0 2B	Typematic	104	71	F0 71	Make only
35	34	F0 34	Typematic	105	84	F0 84	Make only
36	33	F0 33	Typematic	106	7C	F0 7C	Typematic
37	3B	F0 3B	Typematic	108	79	F0 79	Make only
38	42	F0 42	Typematic	110	08	F0 08	Make only
39	4B	F0 4B	Typematic	112	07	F0 07	Make only
40	4C	F0 4C	Typematic	113	0F	F0 0F	Make only
41	52	F0 52	Typematic	114	17	F0 17	Make only
42 **	53	F0 53	Typematic	115	1F	F0 1F	Make only
43	5A	F0 5A	Typematic	116	27	F0 27	Make only
44	12	F0 12	Make/Break	117	2F	F0 2F	Make only
45**	13	F0 13	Typematic	118	37	F0 37	Make only
46	1A	F0 1A	Typematic	119	3F	F0 3F	Make only
47	22	F0 22	Typematic	120	47	F0 47	Make only
48	21	F0 21	Typematic	121	4F	F0 4F	Make only
49	2A	F0 2A	Typematic	122	56	F0 56	Make only
50	32	F0 32	Typematic	123	5E	F0 5E	Make only
51	31	F0 31	Typematic	124	57	F0 57	Make only
52	3A	F0 3A	Typematic	125	5F	F0 5F	Make only
53	41	F0 41	Typematic	126	62	F0 62	Make only
54	49	F0 49	Typematic				

* 101-key keyboard only.
** 102-key keyboard only (non-US).
*** Asian keyboard only.

Table 5-27 lists the range of typematic rate values for the keyboard as set by the 0F3H command. The default values for the keyboard are:

Typematic rate = 10.9 characters per second ± 20%
Delay = 500 milliseconds ± 20%

Table 5-27. Typematic Rate

Bit	Typematic Rate ± 20	Bit	Typematic Rate ± 20
00000	30.0	10000	7.5
00001	26.7	10001	6.7
00010	24.0	10010	6.0
00011	21.8	10011	5.5
00100	20.0	10100	5.0
00101	18.5	10101	4.6
00110	17.1	10110	4.3
00111	16.0	10111	4.0
01000	15.0	11000	3.7
01001	13.3	11001	3.3
01010	12.0	11010	3.0
01011	10.9	11011	2.7
01100	10.0	11100	2.5
01101	9.2	11101	2.3
01110	8.6	11110	2.1
01111	8.0	11111	2.0

The typematic rate (make codes per second) is 1 for each **period**. The period is the interval from one typematic output to the next as determined by this equation:

Period = $(8 + A) \times (2^{B}) \times 0.00417$ seconds. Where:

A = binary value of bits 2, 1, and 0.
B = binary value of bits 4 and 3.

Note that if the keyboard receives a command other than the rate/delay value byte, the execution of 0F3H is halted without change to the existing rate.

Table 5-27 lists the range of typematic rate values for the keyboard as set by the 0F3H command. The default values for the keyboard are:

Typematic rate = 10.9 characters per second ± 20%
Delay = 500 milliseconds ± 20%

Table 5-27. Typematic Rate

Bit	Typematic Rate ± 20	Bit	Typematic Rate ± 20
00000	30.0	10000	7.5
00001	26.7	10001	6.7
00010	24.0	10010	6.0
00011	21.8	10011	5.5
00100	20.0	10100	5.0
00101	18.5	10101	4.6
00110	17.1	10110	4.3
00111	16.0	10111	4.0
01000	15.0	11000	3.7
01001	13.3	11001	3.3
01010	12.0	11010	3.0
01011	10.9	11011	2.7
01100	10.0	11100	2.5
01101	9.2	11101	2.3
01110	8.6	11110	2.1
01111	8.0	11111	2.0

The typematic rate (make codes per second) is 1 for each period. The period is the interval from one typematic output to the next as determined by this equation:

Period = $(8 + A) \times (2^{B}) \times 0.00417$ seconds. Where:

A = binary value of bits 2, 1, and 0.
B = binary value of bits 4 and 3.

Note that if the keyboard receives a command other than the rate/delay value byte, the execution of 0F3H is halted without change to the existing rate.

8042 to STD-BIOS Scancodes and Commands

The keyboard sends scancodes and commands to STD-BIOS driver system . The scancodes/commands are read from the 8042 data port (Input Port 60H). Table 5-28 lists the keyboard codes returned by the keyboard.

Table 5-28. 8042 to STD-BIOS Scancodes and Commands

Code/ Command	Description
00H	OVERRUN. This code indicates that the 16 character keyboard buffer has overflowed.
01H-77H	Keyboard Scancodes. These represent the keys on the 81H-0F7H keyboard. The translations for these scancodes are listed in Table 5-6.
0AAH	The 8042 controller will report this byte when it completes the 8042 controller's Self Test. This test is executed at power-on and, after receiving the Keyboard Command OFFH, reset. Note: any other byte reported at these times indicates failure.
0EEH	ECHO: this code is sent in response to the keyboard ECHO_COMMAND command, OEEH.
0F0H	Break Prefix code. This code is sent to indicate a key break. This code is followed by the scancode of the key being released. This code will be sent only in the AT scancode set mode.
0FAH	ACK. this code is sent to acknowledge receipt of a command (except Echo and Resend).
0FCH	Keyboard Self Test Failure. This code is sent by the keyboard to indicate a failure during the keyboard Self Test (Keyboard Command OFFH).
0FDH	Diagnostic Failure. This code is sent if a keyboard failure is detected.
0FEH	Resend. This code is sent if the keyboard receives an invalid command or detects an error in the transmission.

Logical Keyboard to 8042 Driver Communication

The 8042 acts as an intelligent bi-directional buffer between the logical keyboard driver (Input System) and the INT 09H driver and system software. The INT 09H driver and system software communicate with the 8042 via the command and data ports (I/O addresses 64H and 60H respectively). The 8042 has an additional port (I/O address 68H) which is used by the logical keyboard driver to transfer data and commands to the 8042 without overlapping with the industry standard keyboard commands. Data such as keyboard scancodes and commands are transmitted in this manner. To verify that the command has been read, the software can read the IBF bit in the status register of the controller. The HP specific commands to the 8042 are listed in Table 5-29.

Table 5-29. HP-Specific Commands to the 8042

Keycode Value	Keycode/Command Definition
00H-054H	Industry standard make scancodes. The data byte is put into an 8042 internal scancode buffer, then will loopback the scancode buffer when the 8042's output port is empty.
80H-0D4H	Industry standard break scancodes. The data byte is put into an 8042 internal scancode buffer, then will loopback the scancode buffer when the 8042's output port is empty.
055H-077H	HP-enhanced keyboard make scancodes. The data byte is put into an 8042 internal scancode buffer, then will loopback the scancode buffer when the 8042's output port is empty.
0D5H-0F7H	HP-enhanced keyboard break scancodes. The data byte is put into an 8042 internal scancode buffer, then will loopback the scancode buffer when the 8042's output port is empty.
078H	Reserved
079H	Reserved
07AH	Pass through the next data byte written to output port 068H. The data byte will be put into an 8042 internal scancode buffer, then will loopback the scancode buffer when the 8042's output port is empty.
07BH	Set the RAM Switch to '0'.
07CH	Set the RAM Switch to '1' (Default).
07DH	CRT_OFF: Set the CRT Switch to '0'. Indicates the primary display adapter is a IBM Monochrome/Printer or HP Monochrome Plus adapter.
07EH	CRT_ON: Set the CRT Switch set to '1'. Indicates the primary display adapter is the IBM Color/Graphics or HP Multimode adapter (Default).
07FH	HP Reserved

Table 5-29. HP-Specific Commands to the 8042 (Cont.)

Keycode Value	Keycode/Command Definition
0F8H	ENABLE_AUTOPOLL: Enables the SVC Port request AUTOPOLL_EVENT to be sent to the system. This command allows the CPU to take over the HP-HIL polling function. The AUTOPOLL_EVENT SVC request is made approximately 60 times a second whenever this command is in effect.
0F9H	DISABLE_AUTOPOLL: Disable the AUTOPOLL_EVENT SVC request.
0FAH-0FEH	Reserved
0FFH	KEYBOARD_OVERRUN: This is passed through as any normal keyboard scancode. This command is sent from the 8042 driver to the logical keyboard to the 8042 chip to indicate the logical keyboard's data buffer was overrun.

6

Serial and Parallel I/O

This chapter covers the ROM BIOS support for the system serial and parallel I/O ports. The ROM BIOS supports up to three parallel ports and up to four serial ports.

Note: HP Vectra systems using MS-DOS 3.1 version A.01.04 or greater support three parallel and four serial ports. MS-DOS 3.1 versions less than A.01.04 support three parallel and two serial ports.

Overview

The ROM BIOS provides two STD-BIOS drivers that control the serial (INT 14H) and parallel (INT 17H) ports. The functions in these drivers provide a means of setting communication parameters and transmitting data. These drivers have expanded functionality that provide the programmer with the ability to set higher baud rates and to transfer strings of data. In addition to these drivers, the print screen driver (INT 05H) will be discussed in this chapter.

Serial and Parallel Port Addresses

The STD-BIOS data area contains two tables used by the serial and parallel port drivers. The Serial Base Port Address Table contains the base port addresses for the serial ports. The Parallel Base Port Address Table contains the base port addresses of the parallel ports. The ROM BIOS checks during SYSGEN for the presence of serial and parallel adapter cards at the addresses listed in Table 6-1. When a valid port is found, the base address of that port is placed in the next available entry of the appropriate table. Application programs may add additional parallel ports or serial ports to the port tables. An application program can also replace the values in the table with new ones to support non-standard port addresses. Each table contains space for four entries.

Table 6-1. Serial and Parallel Port Addresses

I/O Address	IRQ	INT	Port	Name
3F8H	4	0CH	0	COM1, AUX
2F8H	3	0BH	1	COM2
3E8H	10	72H	2	COM3
2E8H	11	73H	3	COM4
3BCH	--	--	0	LPT1, PRN
378H	7	0FH	1	LPT2
278H	5	0DH	2	LPT3

Port addresses are added to the base port address tables in the sequence listed in Table 6-1. If the system has only two parallel I/O ports at addresses 378H and 278H, then 378H becomes the first entry in the table (Port 0 - LPT1, PRN) and 278H becomes the second (Port 1 - LPT2). The potential parallel port at 3BCH would not be Port 0 as it is not present in the system.

The functions supported by the serial and parallel port drivers rely on the values contained in the serial base port address table and the parallel base port address table. The ports are referenced by indexes to the tables (port numbers 0-3).

Print Screen Driver

The print screen driver provides a simple method for application programs and system software to print a copy of the screen contents to the system printer (port 0). The ROM BIOS print screen driver will only print the screen if the display adapter is in one of the alphanumeric modes. Support for printing the screen when in graphics modes is provided by the DOS command GRAPHICS.

Polled and Interrupt Driven Operations

Both the serial and parallel ports on the system may be operated in either a polled or interrupt mode. The drivers in the ROM BIOS only support polled operation. Four system interrupts, 0BH, 0CH, 72H and 73H, are reserved for system serial ports. Two system interrupts, 0DH and 0FH, are reserved for system parallel printers. Application programs and system software may use these interrupts to operate the ports in an interrupt mode.

Data Structures

The data structures for the serial port, parallel port, and print screen drivers are located in the STD-BIOS data area. The data structures for each of the drivers are discussed separately.

Serial Port Driver Data Structures

The serial port driver uses two data structures in the STD-BIOS data area; a base port address table, and a timeout counter table. The addresses of these data structures are listed in Table 6-2. The equipment word in the STD-BIOS data area (40:10H), contains the number of serial and parallel ports configured in the system. The equipment byte can be read by the INT 11H equipment determination function.

Table 6-2. Serial Port Data Structures

Port Number	Port Address Table Entry	Timeout Table Entry	Timeout (Default)
0	40:00H	40:7CH	(01H)
1	40:02H	40:7DH	(01H)
2	40:04H	40:7EH	(01H)
3	40:06H	40:7FH	(01H)

Each serial port is comprised of eight I/O addresses. The base address of each block of I/O addresses is stored in the base port address table. For more information, see the *HP Vectra Hardware Technical Reference Manual* (for the HP Vectra ES, QS, or RS personal computers). The table consists of 4 words (8 bytes), one for each of the four possible serial ports. A zero value for any of the words is interpreted by the driver to mean the port is not present.

The second data structure used by the serial port driver is the timeout table. This data structure consists of 4 bytes, one for each of the serial ports. Whenever the driver attempts to read or write data or parameters it reads the status port on the serial port. To prevent an error condition on the serial port from hanging up the system it uses a timeout loop. If a valid status byte cannot be read within the time allotted, the driver will return with a timeout error status code. The length of the timeout is determined by the entries in the timeout table. Each of the four serial ports can be given a different timeout value by an application program.

Parallel Port Driver Data Structures

The parallel port driver uses two data structures that are similar to those used by the serial port driver: the base port address table and timeout counter table. Base port addresses and timeout tables for the parallel port driver are listed in Table 6-3.

Table 6-3. Parallel Port Data Structures

Port Number	Port Address Table Entry	Timeout Table Entry	Timeout (Default)
0	40:08H	40:78H	(14H)
1	40:0AH	40:79H	(14H)
2	40:0CH	40:7AH	(14H)
3	40:0EH	40:7BH	(14H)

Each of the parallel ports occupy four I/O addresses. The base or first address of each is contained in the base address table. A zero value for any of the words is interpreted by the driver to mean the requested parallel port adapter is not present.

The parallel printer port driver checks the status of the port before it outputs a character to determine if the printer is busy. To prevent an error condition on the parallel port from hanging up the system, a timeout loop is used. The length of the timeout is determined by the values stored in the timeout table. The timeout values for each of the parallel ports can be set independently of each other.

Print Screen Driver Data Structures

The print screen driver uses a single byte data structure, located at 0040:0100H (see Appendix B). The print screen driver places a status byte at this location, indicating whether or not a print screen operation is underway. The possible values for this status byte are:

Data	Definition
0	The print screen driver has not been called or it completed the previous operation successfully
1	Printing is in progress.
0FFH	Error occurred during printing.

If this byte indicates a print screen operation is currently in progress, the driver will return. This prevents more than one print screen operation from occurring at the same time.

Serial Port Driver (INT 14H)

The functions supported by the serial port driver can be divided into two groups; those that set and report communication protocol or status, and those that transmit and receive data. The driver supports nine functions. Four of these functions implement the features of the industry standard INT 14H driver. The remaining five functions are EX-BIOS extensions. The ROM BIOS supports several features not found in the industry standard INT 14H driver. Among these features is the ability to select a communication speed of up to 19.2 K baud per second and the support of block (multi-byte) data transfer.

Table 6-4 summarizes each of the Serial Port Driver (INT 14H) functions. It is followed by a description of each function.

Table 6-4. Serial Port Driver Function Code Summary

Function Equate	Definition	Function Value
INT_SERIAL	Serial	14H
F14_INIT	Initialize Serial Port Parameters	00H
F14_XMIT	Send Out One Character	01H
F14_RECV	Receive One Character	02H
F14_STATUS	Get Serial Port Status	03H
F14_INQUIRE	EX-BIOS present	6F00H
F14_EXINIT	Initialize serial port (19.2 Kbaud)	6F01H
F14_PUT_BUFFER	Write a buffer of data	6F02H
F14_GET_BUFFER	Read a buffer of data	6F03H
F14_TRM_BUFFER	Read a buffer of data, terminate on specified condition	6F04H

Serial Port Driver Function Definitions

All of the following functions range check (between 0 and 3 inclusive) the requested port number specified in the DX register. If legal, the function looks up the I/O address contained in the STD-BIOS data area. If the port table entry is non-zero, the port is assumed to exist. If the port table entry is zero the function returns without altering any registers.

F14_INIT (AH = 00H)

The initialize function, F14_INIT, sets the baud rate, number of stop bits, parity and character length of the specified serial port. On return it reports the current contents of the line status register and the modem status register of the specified port.

```
On Entry: AH = F14_INIT (00H)
          AL = Port attribute

            Bit      Data   Definition

          07H-05H    111    9600 baud rate
                     110    4800 baud rate
                     101    2400 baud rate
                     100    1200 baud rate
                     011    600 baud rate
                     010    300 baud rate
                     001    150 baud rate
                     000    110 baud rate
          04H-03H    x0     no parity
                     11     even parity
                     01     odd parity
          02H        0      1 stop bit
                     1      2 stop bits
          01H-00H    00     5 bits
                     01     6 bits
                     10     7 bit character
                     11     8 bit character

          DX = Port number (0, 1, 2, 3)

On Exit: AH = Line status (see Table 6-5)
         AL = Modem status (see Table 6-6)

Registers Altered: AX
```

Table 6-5 defines the Serial Port Line Status.

Table 6-5. Line Status Register Report

Bit	Data	Definition
7	1	Timeout Error (Not applicable on F14_INIT, F14_EXINIT or F14_STATUS)
6	1	Transmit Shift Register Empty
5	1	Transmit Hold Register Empty
4	1	Break Received
3	1	Character Framing Error
2	1	Parity Error
1	1	Overrun Error
0	1	Data Set Ready

Table 6-6 defines the Serial Port Modem Status.

Table 6-6. Modem Status Register Report

Bit	Data	Definition
7	1	Receive Line Signal Detected
6	1	Ring Indicator Line State
5	1	Data Set Ready Line State
4	1	Clear to Send Line State
3	1	Change in Receive Line Detected
2	1	Trailing Edge of Ring Detected
1	1	Change in Data Set Ready
0	1	Change in Clear to Send State

```
Example:
     MOV AH, F14_INIT  ; (AH = 0H)
     MOV AL, 11100111B ; HP LaserJet factory default
                       ; 9600 baud
                       ; No parity
                       ; 2 stop bits
                       ; 8 bit character
                       ; setting
     MOV DX, 0         ; Port 0 specification
     INT INT_SERIAL    ; Call serial driver (INT 14H)
```

F14_XMIT (AH = 01H)

Transmits a data byte on the serial port specified by the DX register. The function enables the REQUEST-TO-SEND and DATA-TERMINAL-READY signals, and then waits on the DATA-SET-READY, CLEAR-TO-SEND, and REGISTER-EMPTY signals until the character is transferred or a timeout occurs.

```
On Entry: AH = F14_XMIT (01H)
          AL = Data byte to be transmitted
          DX = Port number (0, 1, 2, 3)

On Exit: AH = Line status (see Table 6-5)
         AL = Modem status (see Table 6-6)

Registers Altered: AX
```

Example:

```
     MOV AH, F14_XMIT        ; (AH = 02H)
     MOV AL, 'G'             ; ASCII 'G' character to send
     MOV DX, 0               ; Port 0 specification
     INT INT_SERIAL          ; Call serial driver (INT 14H)
     TEST AH, 10000000B      ; Check for error
     JNZ short ERROR_HANDLER
```

F14_RECV (AH = 02H)

This function reads a data byte from the serial port specified by the DX register. The signal DATA-TERMINAL-READY is enabled in the modem control register indicating to the remote device that data can be sent. The modem status register signal DATA-SET-READY and the line status register signal DATA-READY are polled until a data byte is available to read or the timeout count has expired.

```
On Entry: AH = F14_RECV (02H)
          DX = Port number (0, 1, 2, 3)

On Exit: AH = Line status (see Table 6-5)
         AL = If no error: Data byte received

If error: Null character, zero

Registers Altered: AX
```

Example:

```
     MOV AH, F14_RECV        ; (AH = 2)
     MOV DX, 0               ; Port 0 specification
     INT INT_SERIAL          ; Call serial driver (INT 14H)
     TEST AH, 10000000B      ; Check for error
     JNZ short ERROR_RECEIVE
```

F14_STATUS (AH = 03H)

This subfunction returns the status of the serial port specified by the DX register.

```
On Entry: AH = F14_STATUS (03H)
          DX = Port number (0, 1, 2, 3)

On Exit: AH = Line status (see Table 6-5)
         AL = Modem status (see Table 6-6)

Registers Altered: AX
```

F14_INQUIRE (AX = 6F00H)

This function determines whether or not the extended EX-BIOS functions are available. If the EX-BIOS functions are available, the BX register will be set to 4850H (which represents the ASCII characters 'HP').

```
On Entry: AX = F14_INQUIRE (6F00H)
          BX = Any value except 4850H ('HP')

On Exit: BX = 'HP'

Registers Altered: AX, BX
```

Example:

```
     MOV AX, F14_INQUIRE                       ; (AH = 6F00H)
     XOR BX, BX                                ; Clear out BX
     INT INT_SERIAL                            ; Call serial driver (INT 14H)
     CMP BX, 'HP'                              ; Check?
     JNE short ERROR_NO_EXTENDED_FUNCTIONS
```

F14_EXINIT (AX = 6F01H)

This function is similar to the STD-BIOS function, F14_INIT, but provides the ability to set a baud rate beyond 9600.

```
On Entry: AX = F14_EXINIT (6F01H)
          BX = Port attributes

          Bit       Data  Definition

          08H-05H   1000  19200 baud rate
                    0111  9600 baud rate
                    0110  4800 baud rate
                    0101  2400 baud rate
                    0100  1200 baud rate
                    0011  600 baud rate
                    0010  300 baud rate
                    0001  150 baud rate
                    0000  110 baud rate
          04H-03H   x0    no parity
                    11    even parity
                    01    odd parity
          02H       0     1 stop bit
                    1     2 stop bits
          01H-00H   00    undefined
                    01    undefined
                    10    7 bit character
                    11    8 bit character

          DX = Port number (0, 1, 2, 3)

On Exit: AH = Line status (see Table 6-5)
         AL = Modem status (see Table 6-6)

Registers Altered: AX
```

Example:

```
MOV AX, F14_EXINIT           ; (AH = 6F01H)
MOV BX, 0000000100011010B    ; Port attributes
                             ; 19.2 K baud
                             ; parity even
                             ; 1 stop bit
                             ; 7 bit character
MOV DX,1                     ; Port 1 specification
INT INT_SERIAL               ; Call serial driver (INT 14H)
```

F14_PUT_BUFFER (AX = 6F02H)

This function transmits data from a buffer as long as there is data in the data buffer and no error is encountered.

For each data byte transferred, the function enables the REQUEST-TO-SEND and DATA-TERMINAL-READY signals, and then waits on the DATA-SET-READY, CLEAR-TO-SEND, and REGISTER-EMPTY signals until the character is transferred or a timeout occurs. The timeout count is reset for each byte transferred.

```
On Entry: AX = F14_PUT_BUFFER (6F02H)
          CX = number of characters in the data buffer
          DX = Port number (0, 1, 2, 3)
       ES:DI = Pointer to a data buffer of characters

On Exit: AH = Line status (see Table 6-5)
         AL = Modem status (see Table 6-6)

Normal Completion:
         AL = last byte read
         CX = Number of bytes transferred successfully
      ES:DI = Base of data buffer

Error Completion (bit 7 of AH register non-zero):
         CX = Number of bytes transferred successfully
      ES:DI = pointer to next byte to be transferred

Registers Altered: AX, CX, DI, ES
```

Example:

```
STRING DB 'Hello'
END_STRING:
START:
     MOV  AX, seg STRING        ; set pointer to string
     MOV  ES, AX
     MOV  DI, offset STRING
     MOV  AX, F14_PUT_BUFFER    ; (AX =6F02H)
     MOV  CX, END_STRING-STRING ; length of character string
     MOV  DX, 0                 ; Port 0 specification
     INT  INT_SERIAL            ; Call serial driver (INT 14H)
     TEST AH, 10000000B         ; test for errors
     JNZ  short ERROR_PUT_STRING
```

F14__GET__BUFFER (AX = 6F03H)

This function reads characters into the specified data buffer until the buffer is full or a timeout occurs. For each byte, the signal DATA-TERMINAL-READY is enabled in the modem control register, indicating to the remote device that data can be sent. The modem status register signal DATA-SET-READY and the line status register signal DATA-READY are polled until a data byte is available to read or the timeout count has expired.

```
On Entry: AX = F14_GET_BUFFER (6F03H)
          CX = maximum buffer size
          DX = Port number (0, 1, 2, 3)
       ES:DI = Pointer to a data buffer

On Exit: AH = Line status (see Table 6-5)

Normal Completion:
         AL = last byte read
         CX = Number of bytes transferred successfully
      ES:DI = Base of data buffer

Error Completion (bit 7 of AH register non-zero):
         AL = 0, the null byte
         CX = Number of bytes transferred successfully
      ES:DI = pointer to next byte to be transferred

Registers Altered: AX, CX, DI, ES

Example:
IN_BUFFER      DB      512 DUP (20H)
END_BUFFER:
START:
     MOV  AX, seg IN_BUFFER          ; set pointer to string
     MOV  ES, AX
     LEA  DI, offset IN_BUFFER
     MOV  AX, F14_GET_BUFFER         ; (AX = 6F03H)
     MOV  CX, END_BUFFER--IN_BUFFER  ; length of character string
     MOV  DX, 0                      ; Port 0 specification
     INT  INT_SERIAL                 ; Call serial driver (INT 14H)
     TEST AH, 10000000B              ; test for errors
     JNZ  short ERROR_PUT_STRING
```

F14__TRM__BUFFER (AX=6F04H)

This function will read characters into the specified data buffer until any one of the following three conditions occurs:

- The data buffer is filled with characters.
- A character is read which is between the upper bound and the lower bound, inclusive.
- An error or timeout condition is encountered.

For each byte, the signal DATA-TERMINAL-READY is enabled in the modem control register indicating to the remote device that data can be sent. The modem status register signal DATA-SET-READY and the line status register signal DATA-READY are polled until a data byte is available to read or the timeout count has expired. After the data byte is read, it is inspected to see if it lies between the two boundary bytes. If the byte is in between the two bytes, then the transfer is terminated. This function is useful for transferring logical records.

```
On Entry: AX = F14_TRM_BUFFER (6F04H)
          BL = lower bound of termination character
          BH = upper bound of termination character
          CX = maximum buffer size
          DX = Port number (0, 1, 2, 3)
       ES:DI = Pointer to a data buffer

On Exit: AH = Line status (see Table 6-5)

Normal Completion Full Transfer:
         AL = last byte read
         CX = Number of bytes transferred successfully
      ES:DI = Base of data buffer

Normal Completion Terminate Character Detected:
         AL = last byte read (terminate byte)
         CX = Number of bytes transferred successfully
      ES:DI = Base of data buffer

Error Completion (bit 7 of AH register non-zero):
         AL = 0, the null byte
         CX = Number of bytes transferred successfully
      ES:DI = pointer to next byte to be transferred

Registers Altered: AX, CX, DI, ES
```

Example:

```
IN_BUFFER      DB      512 DUP (20H)
END_BUFFER:
START:
     MOV  AX, seg IN_BUFFER         ; set pointer to string
     MOV  ES, AX
     LEA  DI, offset IN_BUFFER
     MOV  AX, F14_TRM_BUFFER        ; (AX =6F04H)
     MOV  CX, END_BUFFER--IN_BUFFER ; length of character string
     MOV  DX, 0                     ; Port 0 specification
     INT  INT_SERIAL                ; Call serial driver (INT 14H)
     TEST AH, 10000000B             ; test for errors
     JNZ  short ERROR_PUT_STRING
     CMP  AL, BL                    ; lower bound?
     JL   NOT_BETWEEN
     CMP  AL, BH                    ; upper bound?
     JG   NOT_BETWEEN
NOT_BETWEEN:
```

Parallel Port Driver (INT 17H)

The parallel port driver provides several functions that support data transfer on the parallel ports and return status. These functions implement the features of the industry standard INT 17H driver and the EX-BIOS extended functions. The EX-BIOS functions implement features not found in the industry standard functions, such as block (multi-byte) data transfer.

Table 6-7 summarizes the Parallel Port Driver (INT 17H) functions. It is followed by a description of each function.

Table 6-7. Parallel Port Driver Function Code Summary

Function Value	Function Equate	Definition
17H	INT_PRINTER	Printer
00H	F17_PUT_CHAR	Send printer one byte
01H	F17_INIT	Initialize printer port
02H	F17_STATUS	Get printer port status
6F00H	F17_INQUIRE	EX-BIOS present
6F02H	F17_PUT_BUFFER	Write a buffer to printer port

Parallel Port Driver Function Definitions

The following functions range check (between 0 and 3, inclusive) the requested port address specified in the DX register. If legal, the function looks up the I/O address contained in the STD-BIOS data area. If the port table entry is non-zero, the port is assumed to exist. If the port table entry is zero, the function returns without altering any registers.

F17_PUT_CHAR (AH = 00H)

This function prints a character on the parallel port. Valid data is set up on the printer interface for at least 900 nanoseconds. If the BUSY signal indicates that the device is busy, it executes an INT 15H function F15_DEV_BUSY. When it returns from F15_DEV_BUSY, the function waits until the BUSY signal indicates the device is not busy. The function generates a 500 nanosecond data strobe and holds the data valid for at least 900 nanoseconds. The function returns with the port status in the AH register.

```
On Entry: AH = F17_PUT_CHAR (00H)
          AL = Data byte to be transmitted
          DX = Port number (0, 1, 2, 3)

On Exit: AH = Printer port status (see Table 6-8)

Registers Altered: AH
```

Table 6-8 defines the parallel printer port status byte.

Table 6-8. Printer Status

Bit	Data	Definition
7	0	Printer Busy
	1	Printer Not Busy
6	0	Not Ready for Data
	1	Data Acknowledged
5	1	Out of Paper
4	0	Printer Offline
	1	Printer On Line (Selected)
3	1	I/O Error
2	x	Not Used
1	x	Not Used
0	1	Printer Error or Timed out

Example:

```
MOV AH, F17_PUT_CHAR  ; (AH = 00H)
MOV AL, 'W'           ; character to print
INT INT_PRINTER       ; Call printer driver (INT 17H)
TEST AH, 00000001B    ; test for error?
JNZ short ERROR_PRINT
```

F17_INIT (AH = 01H)

This function initializes a parallel printer port. It enables the PRINTER-SELECT signal and activates the PRINTER-INITIALIZE signal. The PRINTER-INITIALIZE signal is held active for at least 50 microseconds. The function returns with the printer port status in the AH register.

```
On Entry: AH = F17_INIT (01H)
          DX = Port number (0, 1, 2, 3)

On Exit: AH = Printer port status

Registers Altered: AH
```

Example:

```
MOV AH, F17_INIT  ; (AH = 01H)
INT INT_PRINTER   ; Call printer driver (INT 17H)
TEST AH, 0000001B ; Test for error
```

F17_STATUS (AH = 02H)

This function returns the status of the specified parallel printer port.

```
On Entry: AH = F17_STATUS (02H)
          DX = Port number (0, 1, 2, 3)

On Exit: AH = Printer port status

Registers Altered: AH
```

F17_INQUIRE (AX = 6F00H)

This subfunction determines whether or not the extended EX-BIOS functions are available. If the EX-BIOS functions are available, the BX register will be set to 4850H (which represent the ASCII characters 'HP').

```
On Entry: AX = F17_INQUIRE (6F00H)
          BX = Any value except 4850H ('HP')

On Exit: BX = 'HP'

Registers Altered: AX, BX
```

Example:

```
     MOV  AX, F17_INQUIRE                    ; (AX = 6F00H)
     XOR  BX,BX                              ; Clear out BX
     INT  INT_PRINTER                        ; Call printer driver (INT 17H)
     CMP  BX,'HP'                            ; Check?
     JNE  short ERROR_NO_EXTENDED_FUNCTIONS
```

F17_PUT_BUFFER (AX = 6F02H)

This function transmits data from a buffer as long as there is data in the buffer and no error is encountered. Valid data is set up on the printer interface for at least 900 nanoseconds. If the BUSY signal indicates that the device is busy, it executes an INT 15H function F15_DEV_BUSY. When it returns from F15_DEV_BUSY, the function waits until the BUSY signal indicates the device is not busy. The function generates a 500 nanosecond data strobe and holds the data valid for at least 900 nanoseconds. The function returns with the port status in the AH register.

```
On Entry: AX = F17_PUT_BUFFER (6F02H)
          CX = Number of characters in the data buffer
          DX = Port number (0, 1, 2, 3)
       ES:DI = Pointer to a data buffer of characters

On Exit: AH = Printer port status
```

```
Normal Completion:
         CX = Number of bytes transferred successfully
      ES:DI = Base of data buffer

Error Completion (bit 0 of AH register non-zero):
         CX = Number of bytes transferred successfully
      ES:DI = pointer to next byte to be transferred

Registers Altered: AH, CX, DI, ES
```

Example:

```
STRING      DB      'Hello'
END_STRING:
START:
     MOV  AX, seg STRING          ; set pointer to string
     MOV  ES, AX
     MOV  DI, offset STRING
     MOV  AX, F17_PUT_BUFFER      ; (AX = 6F02H)
     MOV  CX, END_STRING-STRING   ; length of character string
     MOV  DX, 0                   ; Port 0 specification
     INT  INT_PRINTER             ; Call printer driver (INT 17H)
     TEST AH, 00000001B           ; test for errors
     JNZ  short ERROR_PUT_STRING
```

Print Screen Driver (INT 05H)

The print screen driver prints the contents of the screen. Each time an INT 05H instruction is executed, the contents of the screen will be printed on the system printer (Port 0). If a print screen operation is already in progress the driver returns without printing the contents of the screen. The print screen driver does not execute functions in the same manner as the other drivers. It performs a single task, and so there are no functions.

The print screen driver is called by the keyboard driver (INT 9H) when the scancode (06AH) for the <Print Screen> key is detected. In addition, application programs may execute an INT 05H instruction any time a copy of the contents of the screen is desired.

The print screen driver can only print the contents of a screen if the display adapter is in one of its alphanumeric modes.

7

Disc

This chapter discusses the ROM BIOS disc drivers. The disc driver (INT 13H) provides a set of functions that control the disc drives and data transfer between the disc drives and the system.

Overview

The disc driver supports four disc types: standard capacity 5.25-inch flexible discs (360 KB), high-capacity flexible discs (1.2 MB), high capacity 3.5-inch discs (1.44 MB) and hard discs. The structure of the disc driver allows additional drives to be easily integrated into the system.

Physical Drive Numbers

Each drive in the system has a physical drive number. Physical drive numbers for flexible discs start with 0, while physical drive numbers for the hard disc start with 80H. In a typical system configured with one high-capacity flexible disc drive, one standard capacity flexible disc drive, and two hard disc drives, the physical drive numbers would be 0, 1, 80H and 81H respectively.

Flexible Disc Drive Support

The disc driver provides support for both standard and high-capacity flexible disc drives. The disc driver supports dual format operation (i.e., reading and writing both types of flexible discs) in the high-capacity disc drive(s). The flexible disc drives are supported with eleven functions that perform read, write, verify, reset, format, and return status tasks.

Hard Disc Drive Support

The system can be configured with an optional hard disc drive. When an internal hard disc drive is added to the system, the disc driver is "expanded" to include the functions that support the hard disc.

The hard disc BIOS is integrated into the system during SYSGEN (the System Generation process). Early in the SYSGEN process, the software interrupt INT 13H is initialized to point to the flexible disc driver code module.

When an INT 13H is executed the hard disc code is called first. The hard disc code checks the physical drive number specified. If it is a hard disc drive number (greater than or equal to 80H) the function is executed by the hard disc driver code module. If the physical drive number indicates a flexible disc drive (less than 80H), the hard disc code module passes control to the flexible disc driver code module by executing an INT 40H.

External Disc Drives

External disc drives can easily be added to the system. There are two methods for doing this. The external disc can supply BIOS code in an option ROM to enter the system. As an alternative, the system could use a DOS installable device driver.

Discs using installable device drivers can not be used as boot devices, since they are loaded in RAM by the operating system. Further, operating systems other than DOS may not recognize the disc in the system. For more information on installable device drivers consult the *Vectra MS-DOS Programmer's Reference Manual*.

Using the option ROM entry mechanism described in the following section, the external hard disc becomes an integrated part of the system and is treated as if it were an internal drive. The first physical hard disc drive, 80H, can then be used as the system boot device.

Data Structures

There are separate data structures for the hard disc and the flexible disc drivers. The flexible disc has three data structures. The flexible disc parameter table holds information necessary for initializing and supporting the flexible disc controller chip. The flexible disc status table holds information about the status of the previous flexible disc operation. The flexible disc operation table contains various disc operating parameters such as drive status, flexible disc data transfer rate, etc. The hard disc has only one data structure. However, each hard disc driver maintains its own copy. The hard disc parameter table is similar to the flexible disc status table. It contains the physical device characteristics for a particular hard disc attached to the system.

Flexible Disc Operation Table

The flexible disc operation table is located in the STD-BIOS data area starting at memory location 0040:008BH (0048BH). It contains parameters used by the disc driver to perform its functions. Data stored in this table allow the high-capacity drives to read or write either standard or high-capacity flexible discs. The contents of the operating parameter table are listed in Table 7-1. For the Vectra RS system only, support for two additional flexible discs is achieved with a special Flexible Disc Expander card. (If this card is installed, the contents of the operation table are expanded.) See Tables 7-1 and 7-1a.

Table 7-1. Flexible Disc Operation Table

Offset	Length in Bytes	Description
8BH	1	Data transfer rate of previous operation
8FH	1	Drive indicators
90-91H	2	Current media type table for drives 0 and 1
92-93H	2	Work area to generate current media types for drives 0 and 1
94-95H	2	Table of current head positions for drives 0 and 1

For Vectra RS systems with a Flexible Disc Expander card installed, the operation table is expanded to include the following:

Table 7-1a. Expanded Flexible Disc Operation Table

Offset	Length in Bytes	Description
D8H	1	Drive indicators for drive 2 and 3
D9-DAH	2	Current media for drives 2 and 3
DB-DCH	2	Work area to generate current media types for drives 2 and 3
DD-DEH	2	Table of current head positions for drives 2 and 3

Flexible Disc Parameter Table

The flexible disc parameter table contains information that controls the overall operation of the flexible disc controller. This table is pointed to by INT 1EH (0:78H). The parameters used to control the flexible disc controller can be changed by providing a new flexible disc parameter table pointer in INT 1EH (0:78H). This is detailed in Table 7-2.

Table 7-2. Flexible Disc Parameter Table

Offset	Length in Bytes	Description
00H	1	Specify command byte 1: step-rate time and head unload time
01H	1	Specify command byte 2: head load time and DMA (Direct Memory Access) mode
02H	1	Motor wait time
03H	1	Bytes per sector; 0=128, 1=256, 2=512, 3=1024
04H	1	Last sector number on track
05H	1	Read/write gap length between sectors
06H	1	Data length for read/write operations
07H	1	Format gap length between sectors

Table 7-2. Flexible Disc Parameter Table (Cont.)

Offset	Length in Bytes	Description
08H	1	Format filler byte
09H	1	Head settle time after seek command
0AH	1	Motor start time in seconds (1/8 second or 125 ms)

Flexible Disc Status Table

The status table for the internal flexible disc driver begins at memory location 0040:003EH (0043EH) in the STD-BIOS Data Area. The contents of this table are listed in Table 7-3.

Table 7-3. Flexible Disc Status Table

Offset	Length in Bytes	Description
3EH	1	Flag byte
3FH	1	Motor status
40H	1	Motor turn off counter
41H	1	Status of previous flexible disc operation
42H	7	Status bytes returned by the flexible disc controller from the previous operation

Hard Disc Parameter Table

The hard disc drive has a set of parameters which are quite different from the flexible disc. Therefore, the contents of the hard disc parameter table are not the same as its flexible disc counterpart.

Interrupt vector 41H contains the address of the first hard disc table while interrupt vector 46H stores the address of the second hard disc table. The contents of the tables are listed in Table 7-4.

Table 7-4. Hard Disc Parameter Table

Offset	Length in Bytes	Description
00H	2	Total number of cylinders
02H	1	Total number of Read/Write Heads
03H	2	Reserved
05H	2	Starting cylinder for write precompensation
07H	5	Reserved
0CH	2	Cylinder to use as landing zone
0EH	1	Number of sectors per track
0FH	1	Reserved

Disc Driver (INT 13H)

The description of this driver is in two parts: the flexible disc driver functions, and hard disc driver functions.

INT 13H Flexible Disc Driver Functions

Table 7-5 lists each of the INT 13H driver flexible disc functions. All registers not specified in the exit parameters are returned unchanged.

Table 7-5. Flexible Disc Driver Function Code Summary

Function AH	Definition
00H	Reset flexible disc subsystem
01H	Get status from last operation
02H	Read sectors from flexible disc
03H	Write sectors to flexible disc
04H	Read verify sectors on flexible disc
05H	Format a track on flexible disc
06-07H	Reserved
08H	Get drive parameters
09-14H	Reserved
15H	Get DASD (Direct Access Storage Device) type
16H	Get disc change line status
17H	Set DASD type for format
18H	Set media type for format

The the status byte returned in AH for the following functions has the following meaning. For the majority of the functions, the carry flag will be set when AH is non-zero:

AH	Meaning
00H	No errors.
01H	Bad command.
02H	Address mark not found.
03H	Attempt to write on a write protected diskette.
04H	Sector not found.
06H	Media changed.
08H	DMA overrun.
09H	64K boundary violation.
0CH	Media type not found.
10H	Bad CRC detected.
20H	Controller failure.
40H	Seek failure.
80H	Time out.

Flexible Disc Driver Function Definitions

Reset Flexible Disc Subsystem (AH = 00H)

Entry	AH	00H
Exit	AH	Status.

Get Status of Last Operation (AH = 01H)

Entry	AH	01H
Exit	AH	Status.

Read Sectors from Flexible Disc (AH = 02H)

Entry	AH	02H
	AL	Number of sectors to read. (Note 1)
	CL	Starting sector number. (Note 2)
	CH	Cylinder number. (see Note 3)
	DL	Drive number (0 - 3).
	DH	Head number (0 or 1).
	ES:BX	Buffer address.
Exit	AH	Status.
	AL	Number of sectors actually read.

Write Sector to Flexible Disc (AH = 03H)

Entry	AH	03H
	AL	Number of sectors to write (Note 1)
	CL	Starting sector number. (Note 2)
	CH	Cylinder number. (Note 3)
	DL	Drive number (0 - 3).
	DH	Head number (0 or 1).
	ES:BX	Buffer address.
Exit	AH	Status.
	AL	Number of sectors actually written.

Read Verify Sectors on Flexible Disc (AH = 04H)

Entry	AH	04H
	AL	Number of sectors to read verify. (Note 1)
	CL	Starting sector number. (Note 2)
	CH	Cylinder number. (Note 3)
	DL	Drive number (0 - 3).
	DH	Head number (0 or 1).
Exit	AH	Status.
	AL	Number of sectors actually verified.

Format Track (AH = 05H)

Entry	AH	05H
	AL	Sectors per track.
	CH	Cylinder number. (Note 3)
	DL	Drive number (0 - 3).
	DH	Head number (0 or 1).
	ES:BX	Points to a 512 byte buffer containing a table of address fields for the track (C, H, R, N). Where C is the cylinder number, H is the head number, R is the record number and N is the number of bytes per sector (0=128, 1=256, 2=512, 3=1024). There should be as many entries as there are sectors on the track.

For example, to format track 5 head 0 with 9 sectors of 512 bytes each and an interleave factor of 1 the table would look like:

```
;       C    H    R    N    ;Position in track

DB      05H, 00H, 01H, 02H  ;1st.
DB      05H, 00H, 02H, 02H  ;2nd.
DB      05H, 00H, 03H, 02H  ;3rd.
DB      05H, 00H, 04H, 02H  ;4th.
DB      05H, 00H, 05H, 02H  ;5th.
DB      05H, 00H, 06H, 02H  ;6th.
DB      05H, 00H, 07H, 02H  ;7th.
DB      05H, 00H, 08H, 02H  ;8th.
DB      05H, 00H, 09H, 02H  ;9th.
```

The number of sectors per track argument (AL) should be set as follows:

Drive	Media	AL
360	320/360	8/9
1.2	320/360	8/9
1.2	1.2	15
720	720	9
1.44	1.44	18

If the drive can support more than one media type, 1.2 MB for example, then the diskette will be formatted with the lasrgest possible capacity. Use INT 13H, function 17H "Set DASD type" and 18H "Set media type" to set the diskette type to be formatted.

The following parameters in the flexible disc parameter table must be changed before formatting the corresponding media:

Media	Drive	GPL	EOT
320K	360K/1.2MB	50H	8
360K	360K/1.2MB	50H	9
1.2MB	1.2MB	54H	15
720K	1.2MB/1.44MB	50H	9
1.44MB	1.44MB	6CH	18

Where:

GPL	Gap Length for format.
EOT	End Of Track (Last sector on track).

Absoulte address 0:78H contains a pointer to the flexible disc parameter table. GPL is the 8th byte in the table and the EOT is the 5th.

The original parameters must be restored after format is complete.

Get Drive Parameters (AH = 08H)

Entry	AH	08H
	DL	Drive number (0 - 3).
Exit	AX	0
	CL	Sectors per track.
	CH	Total number of cylinders.
	DL	Number of flexible discs in system.
	BL	Drive type as stored in CMOS.
	ES:DI	Address of drive parameter table.

In case of errors such as calling the function with an invalid drive number or the drive type is not known and CMOS is not valid then AX,BX,CX,DX,DI and ES will be set to 0.

Get DASD Type (AH = 15H)

Entry	AH	15H
	DL	Drive number (0 - 3).
Exit	AH	0 = Drive not installed.
		1 = Drive installed, change line not available.
		2 = Drive installed, change line available.
		3 = Reserved

AH is valid only if carry flag is cleared (no errors).

Get Disc Change Line Status (AH = 16H)

Entry	AH	16H
	DL	Drive number (0 - 3).
Exit	AH	0 = Disc change line not active. 6 = Disc change line active.

Set DASD Type for Format (AH = 17H)

Entry	AH	17H
	AL	DASD type to set to: 1 = 320K/360K media in 360K drive. 2 = 360K media in 1.2MB drive. 3 = 1.2MB media in 1.2MB drive. 4 = 720K media in 720K drive.
	DL	Drive number (0 - 3).
Exit	None.	

Set Media Type for Format (AH = 18H)

Entry	AH	18H
	CL	Sectors per track.
	CH	Total number of cylinders.
	DL	Drive number (0 - 3).
Exit	ES:DI	Address of drive parameters table for this Sector per track/Cylinders combination if carry is clear otherwise ES:DI is same as was on entry.
	AH	00h = Sectors per track/Cylinders combination is supported and the ES:DI pointer is valid. 01h = This funciton is not available. 0Ch = Sectors per track/Cylinders combination is not supported.

Note 1: Number of sectors (AL):

Drive	Media	AL
360	320/360	1-8/9
1.2	320/360	1-8/9
1.2	1.2	1-15
720	720	1-9
1.44	1.44	1-18

Note 2: **Sector Number (CL)**:

Drive	Media	AL
360	320/360	1-8/9
1.2	320/360	1-8/9
1.2	1.2	1-15
720	720	1-9
1.44	1.44	1-18

Note 3: **Cylinder number (CH)**:

Drive	Media	CH
360	320/360	0-39
1.2	320/360	0-39
1.2	1.2	0-79
720	720	0-79
1.44	1.44	0-79

INT 13H Hard Disc Driver Functions

Table 7-6. Hard Disc Driver Functions

Function (AH)	Description
00H	Reset hard disc and flexible disc subsystem
01H	Get status from last operation
02H	Read sectors from hard disc
03H	Write sectors to hard disc
04H	Read verify sectors on hard disc
05H	Format a track on hard disc
06-07H	Reserved
08H	Get drive parameters
09H	Set drive parameters
0AH	Read long
0BH	Write long
0CH	Seek
0DH	Alternate hard disc reset
0E-0FH	Reserved
10H	Get drive ready status
11H	Recalibrate drive
12-13H	Reserved
14H	Perform controller diagnostics
15H	Get DASD type

The status byte returned in AH for the following functions has the following meaning. For the majority of the functions, the carry flag will be set when AH is non-zero:

AH	Meaning
00H	No errors.
01H	Bad command.
02H	Address mark not found.
04H	Sector not found.
05H	Reset failure.
07H	Set drive parameters failure.
09H	64K boundary violation on transfer size.
0AH	Bad block flag detected.
10H	Bad ECC detected.
11H	Data was corrected.
20H	Controller failure.
40H	Seek failure.
80H	Time out.
AAH	Drive not ready.
BBH	undefined error occured.
CCH	Write fault.

Hard Disc Driver Function Definitions

Reset Hard and Flexible Disc Subsystem (AH = 00H)

Entry	AH	00H
	DL	Drive number (80H = C:, 81H = D:)
Exit	AH	Status.

Get Status of Last Operation (AH = 01H)

Entry	AH	01H
Exit	AH	Status.

Read Sectors from Hard Disc (AH = 02H)

Entry	AH	02H
	AL	Number of sectors to read. (1-80H)
	CL	Low order 6 bits of CL is the starting sector number. (1-63)
	CH	CH will be combined with the high order 2 bits from CL to form a 10 bit cylinder number with CH being the low order 8 bits. (0-1023)
	DL	Drive number (80H = C: or 81H = D:).
	DH	Head number. (0-15)
	ES:BX	Buffer address.
Exit	AH	Status.

Write Sector to Hard Disc (AH = 03H)

Entry	AH	03H
	AL	Number of sectors to write (1-80H)
	CL	Low order 6 bits of CL is the starting sector number. (1-63)
	CH	CH will be combined with the high order 2 bits from CL to form a 10 bit cylinder number with CH being the low order 8 bits. (0-1023)
	DL	Drive number (80H = C: or 81H = D:).
	DH	Head number. (0-15)
	ES:BX	Buffer address.
Exit	AH	Status.

Read Verify Sectors on Hard Disc (AH = 04H)

Entry	AH	04H
	AL	Number of sectors to read verify. (1-80H)
	CL	Low order 6 bits of CL is the starting sector number. (1-63)
	CH	CH will be combined with the high order 2 bits from CL to form a 10 bit cylinder number with CH being the low order 8 bits. (0-1023)
	DL	Drive number (80H = C: or 81H = D:).
	DH	Head number. (0-15)
Exit	AH	Status.

Format Track (AH = 05H)

```
Entry   AH      05H
        AL      Sectors per track.
        CH      CH will be combined with the high order 2 bits
                from CL to form a 10 bit cylinder number with
                CH being the low order 8 bits. (0-1023)
        DL      Drive number (80H = C: or 81H = D:).
        DH      Head number. (0-15)
        ES:BX   Pointer to an interleave table for the track.
```

For every sector on the track there are two bytes in the table that describe the sector. The first byte is a flag byte that is set to 80H if the sector is to be marked as a bad block otherwise the flag is set to 0. The second byte is the sector number to be given to the sector that this table entry is describing. For example, a table for a track of 17 sectors with interleave factor of 2 and no bad blocks would look like:

```
;       Flag  Sector    Position in track.

DB      00h,  01H       ;1st.
DB      00h,  0AH       ;2nd.
DB      00H,  02H       ;3rd.
DB      00H,  0BH       ;4th.
DB      00H,  03H       ;5th.
DB      00H,  0CH       ;6th.
DB      00H,  04H       ;7th.
DB      00H,  0DH       ;8th.
DB      00H,  05H       ;9th.
DB      00H,  0EH       ;10th.
DB      00H,  06H       ;11th.
DB      00H,  0FH       ;12th.
DB      00H,  07H       ;13th.
DB      00H,  10H       ;14th.
DB      00H,  08H       ;15th.
DB      00H,  11H       ;16th.
DB      00H,  09H       ;17th.
```

Get Drive Parameters (AH = 08H)

```
Entry   AH      08H
        DL      Drive number (80H = C:, 81H = D:).

Exit    AX      0
        CL      Low order 6 bits is the number of sectors
                per track. High order 2 bits are high order
                bits of total number of cylinders.
        CH      Low order 8 bits of cylinder number.
                CH will be combined with the high order
                2 bits from CL to form a 10 bit cylinder number.
        DL      Number of discs in system.
        DH      Maximum head number.
        ES:DI   Address of drive parameter table.
```

Set Drive Parameters (AH = 08H)

Entry	AH	08H
	DL	Drive number (80H = C:, 81H = D:).
Exit	AH	Status. The drive parameters are intialized from the drive parameters pointed to by INT 41H vector for drive C: and INT 46H vector for drive D:.

Read Sectors and ECC from Hard Disc (Read Long) (AH = 0AH)

Entry	AH	0AH
	AL	Number of sectors to read. (1-7FH)
	CL	Low order 6 bits of CL is the starting sector number. (1-63)
	CH	CH will be combined with the high order 2 bits from CL to form a 10 bit cylinder number with CH being the low order 8 bits. (0-1023)
	DL	Drive number (80H = C: or 81H = D:).
	DH	Head number. (0-15)
	ES:BX	Buffer address.
Exit	AH	Status.

The read long operation will transfer 512 bytes of data followed by 4 bytes of ECC for each sector.

Write Sectors and ECC to Hard Disc (Write Long) (AH = 0BH)

Entry	AH	0BH
	AL	Number of sectors to write. (1-7FH)
	CL	Low order 6 bits of CL is the starting sector number. (1-63)
	CH	CH will be combined with the high order 2 bits from CL to form a 10 bit cylinder number with CH being the low order 8 bits. (0-1023)
	DL	Drive number (80H = C: or 81H = D:).
	DH	Head number. (0-15)
	ES:BX	Buffer address.
Exit	AH	Status.

The write long operation will transfer 512 bytes of data followed by 4 bytes of ECC for each sector.

Seek to Specified Cylinder (AH = 0CH)

Entry	AH	0CH
	CL	High order 2 bits are high order 2 bits of the cylinder number.
	CH	CH will be combined with the high order 2 bits from CL to form a 10 bit cylinder number with CH being the low order 8 bits. (0-1023)
	DL	Drive number (80H = C: or 81H = D:).
Exit	AH	Status.

Alternate Disc Reset (AH = 0DH)

Entry	AH	0DH
	DL	Drive number (80H = C: or 81H = D:).
Exit	AH	Status.

The alternate disc reset function is the same as function 00H except that the flexible disc subsystem is not affected.

Test Drive Ready (AH = 10H)

Entry	AH	10H
	DL	Drive number (80H = C: or 81H = D:).
Exit	AH	Status.

Recalibrate Drive (AH = 11H)

Entry	AH	11H
	DL	Drive number (80H = C: or 81H = D:).
Exit	AH	Status.

Controller Diagnostics (AH = 14H)

Entry	AH	14H
Exit	AH	Status.

Get DASD Type (AH = 15H)

Entry	AH	15H
	DL	Drive number (80H = C: or 81H = D:).
Exit	AH	0, Not present. 1, Flexible disc, change line not available. 2, Flexible disc with change line. 3, Hard disc. CX:DX is the number of 512 byte sectors on the media.

8

System Drivers

This chapter contains a description of the drivers which control the system functions. The drivers discussed in previous chapters deal with system peripherals such as the disc drives, keyboard, video display adapter, etc. The drivers covered in this chapter control the system itself.

Overview

The system drivers are designed to provide program access to system operating parameters and to support ROM BIOS drivers. These drivers allow programs to determine the system equipment configuration and amount of memory, provide "hooks" for future multi-tasking capability, control vectors in the HP__VECTOR__TABLE, allocate RAM in the EX-BIOS data area, control system strings, manage CMOS memory, and perform system clock functions. An overview of the capabilities of the drivers in each of these categories follows.

Memory Size And Equipment Determination

The ROM BIOS supports two industry standard drivers that report the current system equipment configuration and memory size. These tasks are supported by the INT 11H and INT 12H drivers, respectively.

The equipment determination driver (INT 11H) returns a word (double word on **Vectra QS and RS series**) that describes the current system configuration. The definition of each bit or group of bits in the word is discussed later in this chapter. The number of printer ports, serial ports, presence of a math coprocessor (80287 or 80387), presence of Weitek math coprocessor (**Vectra RS series only**), initial video display mode and number of flexible disc drives are reported by this driver. The default system configuration is read from a CMOS memory location during power-on. If this information does not match the current configuration, a power-on error message is issued and the current configuration is saved for the INT 11H driver.

The memory size driver (INT 12H) returns a word that indicates the number of 1 KB blocks of system RAM present. The amount of memory reported does not include any extended memory, and is adjusted to exclude the amount of RAM occupied by the EX-BIOS data area. For example, in a system equipped with 640 KB of system RAM using a 4 KB EX-BIOS data area, the amount of memory reported by this driver will be 636 KB. The default amount of memory is read from a word of CMOS memory.

Extended System Support

The extended system support driver (INT 15H) provides support for several advanced system features. It provides "hooks" that allow programs to be written to support multi-tasking at a future date. In addition, it allows data to be transferred to and from extended memory, and allows placing the CPU into its protected mode of operation.

EX-BIOS Driver Support

The V_SYSTEM driver is an EX-BIOS driver that provides support tasks for the EX-BIOS drivers. It contains functions that allocate RAM in the EX-BIOS data area and manipulate HP_VECTOR_TABLE entries.

RAM Allocation

The EX-BIOS data area contains three major data structures: the HP_VECTOR_TABLE, the global data area, and the driver's data area. Within each driver's data area is the driver header, describe record (if applicable), and variable storage area. Each entry in the HP_VECTOR_TABLE is three words long and consists of: Driver's IP, CS, and DS in that order. The HP_ENTRY_CODE (default INT 6FH) loads the appropriate driver's data segment DS and jumps to the address CS:IP.

The global data area is used by system drivers that need to share data. Data structures like the EX-BIOS stack and memory management pointers are maintained here.

The driver data area for each driver is dynamically allocated by the V_SYSTEM driver. Each driver's data area is at its data segment (DS) and is generally composed of a standard header followed by any data particular to the driver. If the driver wishes a data area from the EX-BIOS memory it must follow the allocation process described below.

Space is allocated starting from the base of the global data area toward the top of the HP_VECTOR_TABLE as shown in Figure 8-1. When a driver is initialized, the base address of the last driver data area ("last used DS") is passed to the driver. The driver decrements this value by the number of paragraphs (16 bytes) it needs for its data area, then returns this value as the new "last used DS".

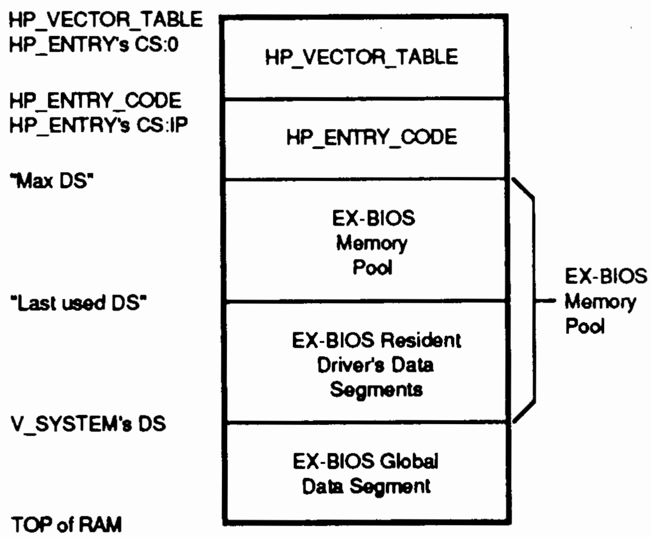


Figure 8-1. Driver Data Area Allocation

If a driver needs a particularly large data area, there might not be enough room. The driver must determine the amount of RAM it requires, then see if that amount is available by comparing its requirements against the amount of RAM available ("last used DS" - "Max DS").

If there is an insufficient amount of RAM available, the driver may increase the amount of RAM allocated to the EX-BIOS data area in the following manner. The memory size stored in CMOS RAM is the amount of physical RAM less the amount occupied by the EX-BIOS Data Area. When the system is booted, the boot code determines the amount of physical memory, then subtracts the "top of memory" stored in CMOS RAM to determine how much space to allocate for the EX-BIOS Data Area. Adjusting the memory size in CMOS RAM downward, then rebooting, will increase the size of the EX-BIOS Data Area and hence the amount of RAM available to the driver. This technique may be used to create an EX-BIOS data area up to 64 KBytes in size. A program listing demonstrating this process follows. (Functions F_RAM_GET, F_RAM_RET, F_CMOS_GET and F_CMOS_RET are described in detail later in this chapter).

Example:

```
     MOV BP, V_SYSTEM              ; How much memory available in
                                   ; EX-BIOS data area?
     MOV AH, F_RAM_GET             ; F_RAM_GET returns:
     CALL SYSCALL                  ;   BX = "last used DS"
                                   ;   DX = "Max DS"
;
     DEC BX                        ; Allocate 3 paragraphs (48 bytes)
     DEC BX                        ; application requires 44 bytes but
     DEC BX                        ; must allocate in full paragraphs
;
     CMP BX, DX                    ; New "last used DS" - "Max DS"
     JA  OK
;
NOT_ENOUGH_RAM:
     MOV BL, 15H                   ; CMOS bytes 16H, 15H contain
                                   ; "top of memory" value
     MOV AH, F_CMOS_GET            ; value (in 1 KB units)
     MOV BP, V_SYSTEM
     CALL SYSCALL                  ; Get least significant byte
;
     DEC AL                        ; Free up 1KB memory for
                                   ; EX-BIOS data area
     PUSHF
     MOV BL, 15H
     MOV AH, F_CMOS_RET
     MOV BP, V_SYSTEM
     CALL SYSCALL                  ; Store new "top of memory" in CMOS
;
     POPF
     JNC RESET_PROCESSOR
;
     MOV BL, 16H                   ; If necessary, decrement most
     MOV AH, F_CMOS_GET            ; significant byte
     MOV BP, V_SYSTEM
     CALL SYSCALL
     DEC AL
     MOV BL, 16H
     MOV AH, F_CMOS_RET
     MOV BP, V_SYSTEM
     CALL SYSCALL

RESET_PROCESSOR:                   ; Reboot system.
                                   ; This time with 1KB more memory
     JMP FAR PTR 0F000H:0FFF0H     ; allocated to the EX-BIOS data area
;
OK:
     MOV BP, V_SYSTEM              ; Set new "last used DS" and "Max DS"
     MOV AH, F_RAM_RET             ; Memory is allocated
     CALL SYSCALL
     .
     .
     .
```

HP_VECTOR_TABLE Manipulation

All drivers in the EX-BIOS code module are accessed through the HP_VECTOR_TABLE. The V_SYSTEM driver provides a set of functions which allows the entries in the HP_VECTOR_TABLE to be set and/or modified. There are nine functions, which represent the permutations of three parameters.

The first parameter determines whether a vector is to be inserted or exchanged with values passed in the CPU registers. Vectors are typically inserted into the HP_VECTOR_TABLE during the boot process, whereas vector exchanges are used to implement driver mapping. For example, the V_QWERTY keyboard translator driver is installed in the HP_VECTOR_TABLE during the boot process. If keyboard scancodes from the QWERTY keypad were to be mapped to a Dvorak translator (**Keyboard/DIN only**), the IP, CS, and DS of the Dvorak translator driver would be exchanged with the existing vector (so the vector could be restored to its original value at a later time).

The second parameter is the vector type. The HP_VECTOR_TABLE has three types of vectors; fixed, reserved, and free. Fixed vectors are those assigned to the default EX-BIOS drivers. The first 51 vectors in the HP_VECTOR_TABLE are fixed. Reserved vectors are set aside for future expansion. There are 24 reserved vectors, which are located at vector addresses 138H through 1C8H inclusive. Free vectors are provided to allow user-supplied drivers to be added to the system.

The final parameter involves the Data Segment (DS) of the driver. Drivers may allocate their data areas from the EX-BIOS data area as explained above, they may provide their own, or use the global data area of the EX-BIOS. The EX-BIOS drivers all use the DS allocation functions, while an external driver (for example, one installed as an MS-DOS device driver) may supply their own data area external to the EX-BIOS data area. Drivers supplying their own DS must pass it as a parameter to V_SYSTEM when the driver has completed initialization.

System String Control

The EX-BIOS provides a centralized and flexible mechanism for accessing and using strings. Each string in the system has a unique index number associated with it. Drivers and application programs can request access to a string via these indices. In addition, functions are available to return the index of a given string, return the next available index, and to add and delete strings from the system.

A string index may be any word value (0--0FFFFH). Certain ranges of indices have predefined meanings or uses. These predefined ranges are listed below.

0--2K	Any index in this range is reserved for string names of EX-BIOS drivers.
2--4K	This range is reserved for strings stored in the ROM-BIOS.
4--32K	This range should be used by application programs to add strings to the system.
32--64K	These indices are reserved for localized strings. Indices within this range are partitioned in the same way as in the lower 32K (i.e., 32--34K for string names of EX-BIOS drivers, etc.).

This index structure provides a powerful tool for localizing application programs. If an application program references messages as string indices, the program can easily be localized by loading a localized set of strings (using a device driver for example), and setting bit 15 of all string indices used.

System strings are grouped into buckets. A bucket is a collection of strings which are grouped together. There is no fixed limit on the number of strings which may be stored in a bucket. However, strings are added and deleted in buckets, not individually. Therefore, strings that are likely to be added or deleted together should be stored in the same bucket.

Each bucket consists of three separate data structures; the bucket header, bucket pointers, and the bucket itself. These components are illustrated in Figure 8-2. The function of each is described below:

Bucket Header - The bucket header is the top level data structure. All bucket headers are linked together in a chain. The first two fields in the header contain the offset and segment of the next bucket header in the chain. If these fields both contain 0FFFFH, then this bucket header is the last in the chain. The highest and lowest string indices contained in the bucket are stored in the next two fields. The following two fields contain the offset and segment of the bucket pointer. Finally, the last field contains the segment of the strings themselves.

Bucket Pointer - The bucket pointer consists of a series of offsets to the strings in the bucket. There must be one offset for every index in the range specified in the bucket header. The actual address of the string is determined by the segment (which is stored in the bucket header) and the offset stored in the bucket pointer. Note that all strings in a bucket must be in the same segment.

Bucket - The bucket contains the actual strings. Each string consists of a byte containing the number of characters in the string, the string itself, and a null byte (00H) which serves as a string terminator.

String control is accomplished through the appropriate functions in the V_SYSTEM driver. These functions provide complete control over system strings.

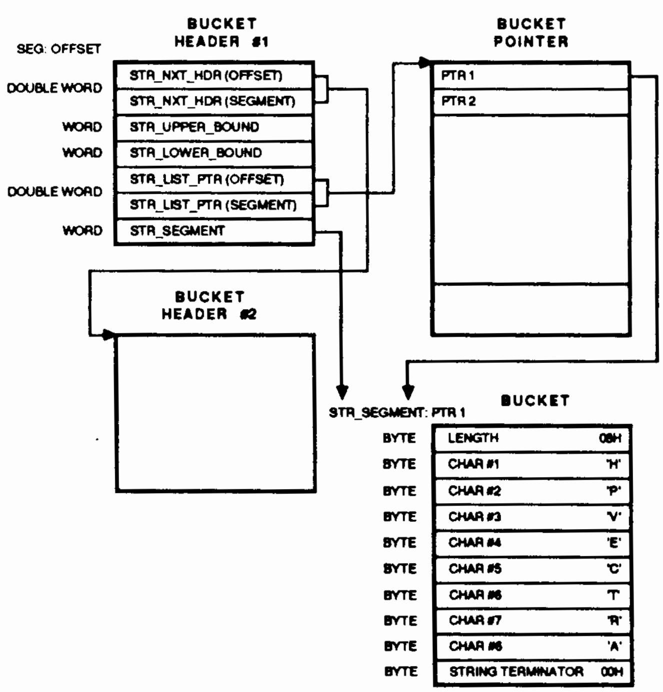


Figure 8-2. System String Data Structures

CMOS Memory Control

The system contains a CMOS Memory/Clock chip that serves as a real-time clock and provides 128 bytes of non-volatile memory storage. The CMOS RAM is used to store system parameters. The contents of the CMOS RAM are listed in Appendix C.

The CMOS Memory/Clock is accessed through two I/O ports. One port selects the clock register or memory byte to access, and the other is a bidirectional data port. There are a total of 128 addresses in the CMOS Memory/Clock chip; the first 14 are the clock registers, while the remaining 114 are the CMOS RAM.

The V_SYSTEM driver contains two functions which support reading and writing data to the CMOS Memory/Clock. These functions provide a simple access to the contents of the chip.

System Clock Functions

The system employs two separate clock systems to keep track of the time and date. The first is the CMOS Memory/Clock. The CMOS clock has a battery back-up which allows it to keep track of the current time when the system is turned off.

The second clock is a software clock. It uses Channel 1 of the 8254 counter/timer chip (refer to the *Vectra Hardware Technical Reference Manual* for additional details). Channel 1 of the 8254 generates a hardware interrupt (IRQ 0) approximately 18.2 times per second. The ROM BIOS keeps time by incrementing a software clock each time the interrupt occurs. The software clock is used by the operating system for such tasks as time and date stamping of files.

The two clocks operate independently except at boot time. During the boot process, the current time and date maintained by the CMOS clock is read and used to initialize the software clock. Changing the value of CMOS clock will not affect the software clock until the system is rebooted.

The STD-BIOS clock driver (INT 1AH) provides a convenient way to read or set the time and date from either of the system clocks. These functions are detailed later in this chapter.

In addition to keeping time, both clocks issue interrupts that call user or application program routines. The software clock interrupt service routine performs an INT 1CH each clock tick. If this vector is modified to point to a user routine, the routine will be called on each clock tick.

The CMOS clock has an "alarm clock" feature. It can be programmed to issue an interrupt at a specified time. The real-time clock hardware issues an INT 4AH each time the alarm timer is done. The interrupt 4AH vector can be modified to point to a user-supplied routine.

Data Structures

The system drivers use several data structures. The data structures for the STD-BIOS system drivers are contained in the STD-BIOS data area and use four data structures. The memory size and equipment determination drivers each use a word; the ROM software clock uses five bytes. These data structures are located at 040:13H, 040:10H, and 040:6CH respectively. The extended system support driver uses 9 bytes starting at location 040:98H. The EX-BIOS drivers are in the EX-BIOS data area and and use the global data area. These data structures are described in detail in Appendix B.

Equipment Determination Driver (INT 11H)

Returns information about the equipment attached to the system.

FOR HP VECTRA ES SERIES COMPUTERS:

On Entry: No Inputs.

On Exit: AX = Word with all equipment information:

Bit	Value	Definition
15, 14		Number of printers attached.
13, 12		Not used.
11, 10, 9		Number of datacomm cards attached.
8		Not used.
7, 6		Number of diskettes attached:
	00	1 drive,
	01	2 drives, only if Bit 0 is also a 1
5, 4		Initial video mode selected:
	00	Other.
	01	40x25 color adapter.
	10	80x25 color adapter.
	11	80x25 monochrome adapter.
3, 2		Not used.
1		Math coprocessor attached.
0	01	Diskette drives attached.

Registers Altered: AX.

FOR HP VECTRA QS AND RS SERIES COMPUTERS:

On Entry: No Inputs.

On Exit: EAX = Double word with all equipment information. (* Indicates for Vectra RS only.)

Bit	Value	Definition
24	0	Weitek 1167* coprocessor not present
24	1	Weitek 1167* coprocessor present
23	0	Weitek 1167* coprocessor addressable by protected mode applications only
23	1	Weitek 1167* coprocessor addressable by real and protected mode applications
15, 14		Number of printers attached.
13, 12		Not used.
11, 10, 9		Number of datacomm cards attached.
8		Not used.
7, 6		Number of diskettes attached:
	00	1 drive,
	01	2 drives, only if Bit 0 is also a 1

5, 4		Initial video mode selected:
	00	Other.
	01	40x25 color adapter.
	10	80x25 color adapter.
	11	80x25 monochrome adapter.
3, 2		Not used.
1		80387 math coprocessor attached.
0	01	Diskette drives attached.

Registers Altered: EAX.

Memory Size Determination Driver (INT 12H)

Returns the amount of RAM found in the system during the power-on and initialization routines.

On Entry: No Inputs.

On Exit: AX = Number of 1KB memory blocks found.

Registers Altered: AX

System Support Driver (INT 15H)

The extended system support driver (INT 15H) provides functions which allow data to be transferred to and from extended memory and allow placing the CPU into its protected mode of operation. These functions are listed in Table 8-1.

Table 8-1. System Support Driver Function Code Summary

Function Value	Function Equate	Definition
	INT_SYSTEM	System Functions Interrupt
0-3		Unsupported
80H	F15_DEVICE_OPEN	Device Open
81H	F15_DEVICE_CLOSE	Device Close
82H	F15_PROG_TERM	Program Termination
83H	F15_WAIT_EVENT	Event Wait
84H	F15_JOYSTICK	Joystick Support
85H	F15_SYS_REQ	System Request Key Pressed
86H	F15_WAIT	Wait Fixed Amount of Time
87H	F15_BLOCK_MOVE	Move Block of Memory to/from Extended Memory
88H	F15_GET_XMEM _SIZE	Get Extended Memory Size
89H	F15_ENTER_PROT	Switch to Protected Mode
90H	F15_DEV_BUSY	Device Busy Hook
91H	F15_INT_COMPLETE	Set Interrupt Completed Flag

System Support Driver Function Definitions

F15_DEVICE_OPEN (AH = 80H)

Open device for I/O. This is a hook for multi-tasking systems. Currently the function just returns.

```
On Entry: AH = F15_DEVICE_OPEN (80H)
          BX = Device Identifier
          CX = Process Identifier

On Exit: No values returned.

Registers Altered: None.
```

F15_DEVICE_CLOSE (AH = 81H)

Close device for I/O. This is a hook for multi-tasking systems. Currently the function just returns.

```
On Entry: AH = F15_DEVICE_CLOSE (81H)
          BX = Device Identifier
          CX = Process Identifier

On Exit: No values returned.

Registers Altered: None
```

F15_PROG_TERM (AH = 82H)

Terminate Program. This is a hook for multi-tasking systems. Currently the function just returns.

```
On Entry: AH = F15_PROG_TERM (82H)
          BX = Device Identifier.
          CX = Process Identifier.

On Exit: No register modified.

Registers Altered: None
```

F15_WAIT_EVENT (AH = 83H)

Allows a process to wait for at least "x" microseconds before it continues. The process is notified that the requested amount of time has elapsed when the high bit at ES:BX is set to "1". If another process is already using this function, the System Support Driver returns with the carry set. If the return status is successful (carry flag is clear), the process should poll the byte at ES:BX until the high bit is set.

```
On Entry: AH = F15_WAIT_EVENT (83H)
          AL = Subfunction:
                 0 = Set the timer with the data passed
                     in ES, BX, CX and DX registers.
                 1 = Cancel the current timer.
       ES:BX = The byte at this address will
               have its high bit set as soon as
               possible after the "x"
               microseconds.
       CX,DX = Minimum number, "x", of microseconds to
               wait before setting the high bit of the
               address above. CX is the most significant
               word.

On Exit: Carry = 1 If there was another process already
                   waiting.
                 0 If the calling process will be notified
                   after the time out.

Registers Altered: AX
```

F15_JOYSTICK (AH = 84H)

Read data from the joystick port.

```
On Entry: AH = F15_JOYSTICK (84H)
          DX = Subfunctions
               0 = Read the switch settings.
               1 = Read resistive inputs.

On Exit: Carry Flag =  0 If no errors
                       1 If invalid DX or no adapter present.

          If DX was 0, AL bits 7..4 contain switch positions.

          If DX was 1, AX = X position of joystick 1
                       BX = Y position of joystick 1
                       CX = X position of joystick 2
                       DX = Y position of joystick 2

Registers Altered: AX, BX, CX, DX
```

Programming Example: To read all the data from the joystick adapter (switches and both joysticks).

```
     MOV AH, F15_JOYSTICK ; Function 84H
     MOV DX, 00           ; Read the switch settings first
     INT INT_SYSTEM       ; Int 15H
     JC HANDLE_ERRORS
     MOV SWITCH_STATE,AL  ; Save the state of the switches
                          ; Bits 7..4 in AL.
     MOV AH, F15_JOYSTICK ; Call it again for joystick info
     MOV DX, 01
     INT INT_SYSTEM
     JC HANDLE_ERRORS
     MOV STICK1_X, AX     ; Save x and y position for both
     MOV STICK1_Y, BX     ; joysticks.
     MOV STICK2_X, CX
     MOV STICK2_Y, DX
     .
     .                    ; Continues normally here
     .
HANDLE_ERRORS:
     .
     .                    ; Error handler here
     .
```

F15_SYS_REQ (AH = 85H)

This subfunction gets called by the keyboard interrupt handler (INT 9H) whenever the user presses the <System request> key. Currently the routine just returns, but an application can trap this function to detect when the user presses this key.

```
On Entry: AH = F15_SYS_REQ (85H)
          AL = 00, If user pressed the <System request> key down (make).
               01, If user let go of the <System request> key (break).
```

On Exit: No values returned.

Registers Altered: None.

Example: Link into the current <System request> handler so that it prints "HELLO" everytime the <System request> key is pressed.

```
INITIALIZATION_CODE:
     MOV AH, 35H              ; Get the old INT 15H
     MOV AL, INT_SYSTEM       ; Get CS:IP of INT 15H
     INT 21H                  ; This MS-DOS Int does the work
     MOV OLD_SEG, ES
     MOV OLD_OFFSET, BX
     MOV AH,25H               ; Replace old INT 15H
     MOV AL, INT_SYSTEM       ; with our routine
     PUSH CS
     POP DS
```

```
     MOV DX, offset OUR_INT15
     INT 21H                     ; This MS-DOS Int does the work
          .
          .
          .
OUR_INT15:
     CMP AH, F15_SYS_REQ         ; See if it is function 85H?
     JNE DO_OLD_INT
     PUSHA
     PUSH ES
     MOV AX, F10_WRS_01          ; Yes, call video "write string"
     MOV BL, 07                  ; function 1301H to write the
     MOV CX, 05                  ; string "HELLO"
     MOV BH, 00                  ; page 0
     MOV DX, 00                  ; row 0, column 0
     PUSH CS
     POP ES
     MOV BP, Offset HELLO_STR
     INT INT_VIDEO               ; Video function interrupt 10H
     POP ES
     POPA
     IRET
DO_OLD_INT:
     PUSH OLD_OFFSET             ; No, just go to regular routine.
     PUSH OLD_SEG
     RET
HELLO_STR DB "HELLO"
```

F15_WAIT (AH = 86H)

Calling this function causes a wait of the specified number of microseconds (CX, DX) before returning to the caller.

```
On Entry: AH = F15_WAIT (86H)
       CX,DX = Number of microseconds to wait.
               CX is the most significant word.

On Exit: Carry = 1, Some other process already
                    waiting. So could not wait.
         Carry = 0, Waited the amount of microseconds
                    specified in the CX,DX register pair.

Registers Altered: None.
```

Example: Wait 10 milliseconds in a procedure.

```
     MOV AH, F15_WAIT          ; 86H function
     MOV CX, 0                 ; 10 * 1000 microseconds
     MOV DX, 10000             ; = 10 milliseconds
     INT INT_SYSTEM            ; INT 15H
     JC HANDLE_ERRORS
     .
     .                         ; At least 10 milliseconds have elapsed
HANDLE_ ERRORS:
     .                         ; Do what's appropriate here.
```

F15__BLOCK__MOVE (AH = 87H)

Moves a block of memory from one location to another anywhere in the addressing space of the CPU. The number of words to move is passed in CX and the source and destination tables pointers are passed in a Global Descriptor Table (GDT) pointed to by ES:SI. The following data structure describes a sample GDT:

```
ADDRESS_DATA STRUC
RESERVED_GDT DB   8 DUP (?) ; Descriptor used during move
CALLERS_GDT  DB   8 DUP (?) ; Caller's GDTs during move
SOURCE_GDT   DB   8 DUP (?) ; GDT describing source
DEST_GDT     DB   8 DUP (?) ; GDT describing destination
BIOS_GDT     DB   8 DUP (?) ; GDT of the BIOS routines
STACK_GDT    DB   8 DUP (?) ; Stack's GDT.
ADDRESS_DATA ENDS
```

The eight-byte descriptor for source or destination has the following format:

```
SAMPLE_GDT     STRUC
SEG_LIMIT      DW   ?  ; Segment Limit
LOW_WORD       DW   ?  ; Low word of 24-bit address
HIGH_BYTE      DW   ?  ; High byte of 24-bit address
ACCESS_RIGHT   DW   ?  ; Segment access rights  should always be 93H
RESERVED_WORD  DW   ?  ; Reserved.
SAMPLE_GDT     ENDS
```

```
On Entry: AH = F15_BLOCK_MOVE (87H)
       ES:SI = Pointer to descriptor tables.
          CX = Number of words to move.

On Exit: AH = Return Status:
              0, If successful.
              1, If RAM parity error.
              2, If exception interrupt error.
              3, If gate address line 20 failed.

         Carry Flag = 1, If failure.
         Zero Flag = 1, If successful.

Registers Altered: AX
```

Example: Move the 16KB video buffer to the procedure's buffer.

```
     MOV  SI, offset DEST               ; Load table with 24 bit
                                        ; destination address:
     MOV  BX, seg BUFFER                ; Isolate high nibble of segment
     AND  BX, 0F000H
     SHR  BX, 12
     MOV  AX, seg BUFFER                ; isolate rest of segment
     SHL  AX, 4
     ADD  AX, offset BUFFER             ; and form 24-bit address
     JNC  SKIP_INC
     INC  BX
SKIP_INC:
     MOV  BYTE PTR HIGH_BYTE[SI], BL
     MOV  WORD PTR LOW_WORD[SI], AX
     LES  SI, ACTUAL_TABLE
     MOV  CX, 8192                      ; Number of words to move
     MOV  AH, F15_MOVE_BLOCK            ; Function 87H.
     INT  INT_SYSTEM                    ; Int 15H
     JC   HANDLE_ERRORS
     JNE  HANDLE_ERRORS
     .                                  ; Continue if
                                        ; everything OKAY
HANDLE_ERRORS:
     .                                  ; Do Error processing here
                                        ; Actual Table of pointers
                                        ; passed to the routines. They
                                        ; use the Global descriptor
                                        ; structure described above.
ACTUAL_TABLE:
RESERVED  SAMPLE_GDT <0,0,0,0,0>
CALLERS   SAMPLE_GDT <0,0,0,0,0>
SOURCE    SAMPLE_GDT <16384,8000H,0BH,93H,0>
DEST      SAMPLE_GDT <16384,0,0,93H,0> ; The high byte
                                        ; and low word will be
                                        ; loaded in the code
BIOS      SAMPLE_GDT <0,0,0,0,0>
STACK     SAMPLE_GDT <0,0,0,0,0>
BUFFER    DB         16384 DUP (?)
                                        ; Actual destination buffer
```

F15_GET_XMEM_SIZE (AH = 88H)

Determine how much RAM there is above the first one megabyte of memory.

On Entry: AH = F15_GET_XMEM_SIZE (88H)

On Exit: AX = Total number of 1KB blocks above one megabyte.

Registers Altered: AX.

F15_ENTER_PROT (AH = 89H)

Allows a routine to enter protected mode. When the BIOS function has executed, the processor will be in protected mode and the routine specified will be called. The calling program must create a set of descriptor tables as follows:

Dummy Descriptor Table:	Initialize to zero.
Global Descriptor Table:	Load program dependent values.
Interrupt Descriptor Table:	Load program dependent values.
Data segment Descriptor:	Load program dependent values.
Extra segment Descriptor:	Load program dependent values.
Stack segment Descriptor:	Load program dependent values.
Code segment Descriptor:	Load program dependent values.
BIOS Descriptor Table:	Initialize to zero.

When calling this function, the user should be aware that:

1. The BIOS functions are not available.
2. The interrupt tables must be moved to avoid conflict with the CPU interrupt vectors.
3. The user loaded descriptor tables must not overlap with the BIOS's descriptor tables.

Upon return from protected mode the system BIOS will return control to the return point specified at 40H:67H. The user should recover the stack and continue.

There are a few points of caution that should be observed:

1. Any code which is expected to run mixed mode, that is both protected mode and real mode, must not make any far references, including far calls.
2. Also, any return addresses put on the stack must have been generated in the same mode in which the return code executes, or else they must be near returns.
3. The system address line A20 must be forced to 0 when the system is operating in real mode. This task is performed by the 8042 controller. When the system enters protected mode, A20 must be released, and when it enters real mode it must be forced to 0 again. It is the program's responsibility to issue the appropriate command to the 8042 controller before changing modes (see Chapter 5).

```
On Entry: AH = F15_ENTER_PROT (89H)
          BH = Offset into interrupt table where interrupts
               coming from the Master 8259 will go
               (Interrupt level 1).
          BL = Offset into interrupt table where interrupts
               coming from the slave 8259 will go (Interrupt level 2)
       ES:SI = Pointer to a set of descriptor tables.
               The following descriptors must be passed
               by the calling routine:
               Dummy Descriptor (DUMMY),
               Global Descriptor Table (GDT),
               Interrupt Descriptor Table (IDT),
               Data Segment Descriptor Table (DS),
               Extra Segment Descriptor Table (ES),
               Stack Segment Descriptor Table (SS),
               Code Segment Descriptor Table (CS) and
               BIOS Descriptor Table (BIOS).

On Exit: AH = 0, If successfully entered Protected Mode.

Registers Altered: All.
```

Example: To enter protected mode and start executing the routine PROTECTED.

```
; Load up descriptor tables with appropriate values. See the
; iAPX 80286 or 80386 Programmer's Reference Manual for details.
;
     .
     .
     .
;
; Load registers for calling INT 15H function.
;
     MOV AH, F15_ENTER_PROT ; Enter protected mode function 89H
; Offset for 8259's must be greater than 32
; because CPU uses the first 32 interrupts vectors.
     MOV BH, 40             ; New offset for master 8259.
     MOV BL, 48             ; New offset for slave 8259.
     MOV ES, seg GLOBAL_TABLE ; Table of descriptors.
     MOV SI, offset GLOBAL_TABLE
     INT INT_SYSTEM           ; Int 15H

PROTECTED:
;
; Code starts executing here after call to INT 15H
; sets up CS_DT to point to PROTECTED label.
;
; Descriptor tables needed for this function
; call. The entries marked by 'F' must be
; filled in by the user. Those marked with
; '0' are filled by INT 15H. For a definition
```

```
; of the SAMPLE_GDT structure see the
; F15_BLOCK_MOVE example. For information as
; to how to fill this table see the iAPX 80286
; or 80386 Programmer's Reference Manual.
;
GLOBAL_TABLE:
RESERVED    SAMPLE_GDT <0,0,0,0,0>
GLBL_DT     SAMPLE_GDT <F,F,F,F,F>
IDT_DT      SAMPLE_GDT <F,F,F,F,F>
DS_DT       SAMPLE_GDT <F,F,F,F,F>
ES_DT       SAMPLE_GDT <F,F,F,F,F>
SS_DT       SAMPLE_GDT <F,F,F,F,F>
CS_DT       SAMPLE_GDT <F,F,F,F,F>
BIOS_DT     SAMPLE_GDT <0,0,0,0,0>
```

F15_DEV_BUSY (AH = 90H)

Device busy function. This is a "hook" for multi-tasking systems. Currently the function just clears the Carry flag and returns.

```
On Entry: AH = F15_DEV_BUSY (90H)
          AL = Device Type:
               0 thru 7FH = Device can not be shared.
                   The operating system handling this "hook"
                   must serialize access to this device.
               80H thru 0BFH = Device can be shared among
                   multiple processes. The operating system
                   handling this "hook" must use the ES:BX
                   registers to distinguish between calls.
               0C0H thru 0FFH = Devices of this type must wait
                   for a fixed amount of time. This amount of
                   time is device dependant. Control should be
                   returned to the device after the fixed amount time
          List of Device Types:
               00H = Disc, timeout required
               01H = Diskette, timeout required
               02H = Keyboard, no timeout required
               80H = Network, no timeout required
              0FDH = Start diskette motor, timeout required
              0FEH = Printer, timeout required.

On Exit: No values returned.

Registers Altered: None.
```

F15_INT_COMPLETE (AH = 91H)

Signals interrupt completed. This is a "hook" for multitasking systems. Currently the function does an IRET.

```
On Entry: AH = F15_INT_COMPLETE (91H)
          AL = Device Type, see list of previous function.

On Exit: No registers used.

Registers Altered: None.
```

Time and Date Driver (INT 1AH)

Table 8-2 describes functions provided by the BIOS to manage the CMOS clock and the software clock.

Table 8-2. Time and Date Driver Function Code Summary

Function Value	Function Equate	Definition
	INT_CLOCK	Time and date
00H	F1A_RD_CLK_CNT	Read current clock count
01H	F1A_SET_CLK_CNT	Set current clock count
02H	F1A_GET_RTC	Read real-time clock
03H	F1A_SET_RTC	Set real-time clock
04H	F1A_GET_DATE	Read date from real-time clock
05H	F1A_SET_DATE	Set date in real-time clock
06H	F1A_SET_ALARM	Set alarm
07H	F1A_RESET_ALARM	Reset alarm

Time and Date Driver Function Definitions

F1A_RD_CLK_CNT (AH = 00H)

Reads the current setting of the software clock. There are 18.2 counts per second.

```
On Entry: AH = F1A_RD_CLK_CNT (00H)

On Exit: AL = Zero if the timer has not overflowed (not passed 24 hours
              since the last read). Nonzero if time has overflowed.
         CX = High word of the count. (There are 18.2 counts per second).
         DX = Low word of count.

Registers Altered: AX, CX, DX
```

F1A_SET_CLK_CNT (AH = 01H)

Sets the count in the software clock. And resets the 24 hour overflow bit.

```
On Entry: AH = F1A_SET_CLK_CNT (01H)
          CX = High word of Count.
          DX = Low word of Count.
```

On Exit: No values returned.

Registers Altered: None

F1A_GET_RTC (AH = 02H)

Gets the time from the real-time clock.

On Entry: AH = F1A_GET_RTC (02H)

```
On Exit: CH = Hours in BCD.
         CL = Minutes in BCD.
         DH = Seconds in BCD.
         Carry flag = 1 if real-time clock is not operating.
```

Registers Altered: AH, CX, DH

F1A_SET_RTC (AH = 03H)

Sets the time of the real-time clock.

```
On Entry: AH = F1A_SET_RTC (03H)
          CH = Hours in BCD.
          CL = Minutes in BCD.
          DH = Seconds in BCD.
          DL = 1 if daylight savings time
               option.  0 otherwise.
```

On Exit: No values returned.

Registers Altered: AH.

F1A_GET_DATE (AH = 04H)

Gets the date from the real-time clock.

On Entry: AH = F1A_GET_DATE (04H)

```
On Exit: CH = 19 if 20th century or 20 if 21st century.
         CL = Year in BCD.
         DH = Month in BCD.
         DL = Day in BCD.
         Carry flag set if the real-time clock not operating.

Register Altered: AH, CX, DX.
```

F1A_SET_DATE (AH = 05H)

Sets the date of the real-time clock.

```
On Entry: AH = F1A_SET_DATE (05H)
          CH = 19 if 20th century or 20 if 21st century.
          CL = Year in BCD.
          DH = Month in BCD.
          DL = Day in BCD.

On Exit: No values returned.

Registers Altered: AH.
```

F1A_SET_ALARM (AH = 06H)

Sets the alarm to generate an INT 4AH when the specified amount of time has elapsed. The user must place an appropriate interrupt handling routine in the INT 4AH vector.

```
On Entry: AH = F1A_SET_ALARM (06H)
          CH = Hours in BCD.
          CL = Minutes in BCD.
          DH = Seconds in BCD.

On Exit: Carry flag = 1 if the real-time clock is not operating
                      or the alarm is already set.

Registers Altered: AH.
```

F1A_RESET_ALARM (AH = 07H)

Clears the current alarm if any was set.

```
On Entry: AH = F1A_RESET_ALARM (07H)

On Exit: No values returned.

Registers Altered: AH.
```

V__SCOPY Driver (BP = 0000H)

This driver does an IRET for all function calls.

V__DOLITTLE Driver (BP = 0006H)

This driver does an IRET for all function calls.

V__PNULL Driver (BP = 000CH)

This driver loads AH with RS__SUCCESSFUL and does an IRET for all function calls.

V__SYSTEM Driver (BP = 0012H)

Table 8-3 summarizes the V__SYSTEM driver Functions. A more detailed description follows the table.

Table 8-3. V__SYSTEM Driver Function Code Summary

Func. Value	Function Equate	Definition
	V__SYSTEM	System Management Functions
00	F__ISR	Interrupt service routine (unsupported)
02	F__SYSTEM	Standard Driver Functions
0200	F__SF__INIT	System initialization
04	F__INS__BASEHPVT	Returns HP__VECTOR__TABLE segment
06	F__INS__XCHGFIX	Exchanges fixed table entries
08	F__INS__XCHGRSVD	Sets next "reserved" entry in table
0A	F__INS__XCHGFREE	Sets next "free" entry in table
0C	F__INS__FIXOWNDS	Install fixed vector, user supplies DS
0E	F__INS__FIXGETDS	Install fixed vector, system supplies DS
10	F__INS__FIXGLBDS	Install fixed vector, DS set to global data area
12	F__INS__FREEOWNDS	Install next free vector, user supplies DS
14	F__INS__FREEGETDS	Install next free vector, system supplies DS
16	F__INS__FREEGLBDS	Install next free vector, DS set to global data area
18	F__INS__FIND	Search for matching device header
1E	F__RAM__GET	Get EX-BIOS memory pool address and size

Table 8-3. V_SYSTEM Driver Function Code Summary (Cont.)

Func. Value	Function Equate	Definition
20	F_RAM_RET	Set memory pool address and size
22	F_CMOS_GET	Read and verify CMOS memory
24	F_CMOS_RET	Write to CMOS memory
2A	F_YIELD	Just returns
2C		Reserved
2E		Reserved
30	F_SND_CLICK _ENABLE	Enable keyclick
32	F_SND_CLICK _DISABLE	Disable keyclick (Default)
34	F_SND_CLICK	Execute keyclick if enabled
36	F_SND_BEEP _ENABLE	Enables beep
38	F_SND_BEEP _DISABLE	Disables beep
3A	F_SND_BEEP	Beeps if enabled
3C	F_SND_SET _BEEP	Sets beep frequency
3E	F_SND_TONE	Produce tone, user supplied duration and frequency
40	F_STR_GET _FREE_INDEX	Return next free string index
42	F_STR_DEL _BUCKET	Delete bucket string list
44	F_STR_PUT _BUCKET	Add bucket to current string list
46	F_STR_GET _STRING	Search the list for index, return string
48	F_STR_GET _INDEX	Search list for a string, return index

Registers Altered: AH, DS, BP, ES

Example: Get the Base address of the HP_VECTOR_TABLE.

```
MOV  BP, V_SYSTEM              ; HP vector (12H).
MOV  AH, F_INS_BASEHPVT        ; function 04H
PUSH DS                        ; EX-BIOS destroys DS
CALL SYSCALL                   ; Int for EX-BIOS (default 6FH)
MOV  AX, DS
POP  DS                        ; Restore DS
PUSH
MOV  GLOBAL_DATA_AREA, AX
MOV  AX, ES
MOV  VECTOR_TABLE_SEGMENT, AX
```

The value returned in ES is the segment address of the HP_VECTOR_TABLE and the value returned in the DS register is the segment address of the EX-BIOS global data area.

V_SYSTEM Driver Function Definitions

F_ISR (AH = 00H)

Logical interrupt service routine. Currently, it loads AH with RS_UNSUPPORTED and does an IRET.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_ISR (00H)

On Exit: AH = RS_UNSUPPORTED (02H)

Registers Altered: AH, BP, DS
```

F_SF_INIT (AX = 0200H)

System functions routines. The only function supported is SF_INIT (00H). The rest of the routines return with a status of RS_UNSUPPORTED in AH.

The SF_INIT routine sets up DS and initializes all the variables in the EX-BIOS global data area.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)

On Exit: AH = Return Status Code
         BX = DS of EX-BIOS global data area

Registers Altered: AH, BX, DS, BP
```

F_INS_BASEHPVT (04H)

Reports the segment where the HP_VECTOR_TABLE is located. This function can only be called after the V_SYSTEM driver has been initialized.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_INS_BASEHPVT (04H)

On Exit: AH = Return Status Code
         ES = Segment address of HP_VECTOR_TABLE.
         DS = Segment of EX-BIOS global data area
```

F_INS_XCHGFIX (AH = 06H)

Exchanges the values in the registers for a particular entry in the HP_VECTOR_TABLE. This function can be used to replace an existing vector at a fixed location without initialization.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_INS_XCHGFIX (06H)
          BX = Vector address
          DX = DS to be exchanged
       ES:DI = CS:IP to be exchanged

On Exit: AH = Return Status Code
              0 = RS_SUCCESSFUL
         DX = DS from table
      ES:DI = CS:IP from table

Registers Altered: AH, BP, DS, ES, DI, DX
```

Example: Replace the EX-BIOS V_SVIDEO vector (54H).

```
MOV BP, V_SYSTEM                ; HP vector 12H.
MOV AH, F_INS_XCHGFIX           ; Function 06H
MOV BX, V_SVIDEO                ; HP vector 54H
MOV DI, CS                      ; Get CS, IP and DS of new
MOV ES, DI                      ; video routines.
MOV DI, offset NEW_VIDEO_ROUTINE
MOV DX, DS
PUSH DS                         ; EX-BIOS Destroys DS
CALL SYSCALL                    ; Int for EX-BIOS (default 6FH)
POP DS
MOV OLD_CS, ES                  ; Save old CS, IP and DS
MOV OLD_IP, DI                  ; just in case we need to
MOV OLD_DS, DX                  ; put them back
```

F_INS_XCHGRSVD (AH = 08H)

Exchanges the values in the registers for the next reserved entry in the HP_VECTOR_TABLE. If a reserved vector is not available, the function returns the RS_NO_VECTOR error code.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_INS_XCHGRSVD (08H)
          DX = DS to be exchanged
       ES:DI = CS:IP to be exchanged

On Exit: AH = Return Status Code
              0    = RS_SUCCESSFUL
              0F6H = RS_NO_VECTOR
         BX = Vector address
         DX = DS from table
      ES:DI = CS:IP to be exchanged

Registers Altered: AH, BP, DS, BX, ES, DI, DX
```

F__INS__XCHGFREE (AH = 0AH)

Exchanges the values in the registers for the next free entry in the HP__VECTOR__TABLE. If a free vector is not available, the function returns the RS__NO__VECTOR error code.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_INS_XCHGFREE (0AH)
          DX = DS to be exchanged
       ES:DI = CS:IP to be exchanged

On Exit: AH = Return Status Code
              0    = RS_SUCCESSFUL
              0F6H = RS_NO_VECTOR
         BX = Vector address
         DX = DS from table
      ES:DI = CS:IP to be exchanged

Registers Altered: AH, BP, DS, BX, ES, DI, DX
```

F__INS__FIXOWNDS (AH = 0CH)

Installs a given vector entry in the HP__VECTOR__TABLE and calls it with an SF__INIT function. Upon returning from initialization, the routine returns its data segment in the BX register.

WARNING

If the SF__INIT function returns with an error code of RS__FAIL (0FEH), the power-on self test sequence will be executed.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_INS_FIXOWNDS (0CH)
          BX = Vector address to be installed
       ES:DI = CS:IP of the device

On Exit: AH = Return Status Code
              0 = RS_SUCCESSFUL

Registers Altered: AH, BP, DS
```

F__INS__FIXGETDS (AH = 0EH)

Installs a given vector entry in the HP__VECTOR__TABLE and calls it with an SF__INIT function. This function should be used if the driver needs EX-BIOS RAM for its data segment. F__INS__FIXGETDS calls the routine to initialize with the "last used DS" in the BX register. The routine's initialization code decrements the "last used DS" value and returns to F__INS__FIXGETDS with this new value.

WARNING

If the SF__INIT function returns with an error code of RS__FAIL (0FEH), the power-on self test sequence will be executed.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_INS_FIXGETDS (0EH)
          BX = Vector address to be installed
       ES:DI = CS:IP of the routine

On Exit: AH = Return Status Code
              0 = RS_SUCCESSFUL

Registers Altered: AH, BP, DS
```


F__INS__FIXGLBDS (AH = 10H)

Installs a given vector entry in the HP__VECTOR__TABLE and calls it with an SF__INIT function. When F__INS__FIXGLBDS calls the initialization routine it passes the data segment of the EX-BIOS global data area in the BX register.

WARNING

If the SF__INIT function returns with an error code of RS__FAIL (0FEH), the power-on self test sequence will be executed.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_INS_FIXGLBDS (10H)
          BX = Vector address to be installed
       ES:DI = CS:IP of the routine

On Exit: AH = Return Status Code
              0 = RS_SUCCESSFUL

Registers Altered: AH, BP, DS
```

F_INS_FREEOWNDS (AH = 12H)

Installs a vector in the next free entry of the HP_VECTOR_TABLE and calls it with an SF_INIT function. Upon returning from initialization, the routine returns its DS in the BX register.

WARNING

If the SF_INIT function returns with an error code of RS_FAIL (0FEH), the power-on self test sequence will be executed.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_INS_FREEOWNDS (12H)
          BX = Vector address to be installed
       ES:DI = CS:IP of the device

On Exit: AH = Return Status Code
              0 = RS_SUCCESSFUL

Registers Altered: AH, BP, DS
```

F_INS_FREEGETDS (AH = 14H)

Installs a vector in the next free entry of the HP_VECTOR_TABLE and calls it with an SF_INIT function. This function is used if the driver needs EX-BIOS RAM for its data segment. F_INS_FREEGETDS calls the routine to initialize with the "last used DS" in the BX register. The routine's initialization code decrements the "last used DS" value and returns it to F_INS_FREEGETDS.

WARNING

If the SF_INIT function returns with an error code of RS_FAIL (0FEH) the power-on self test sequence will be executed.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_INS_FREEGETDS (14H)
       ES:DI = CS:IP of the routine

On Exit: AH = Return Status Code
              0 = RS_SUCCESSFUL

Registers Altered: AH, BP, DS
```

Example: Install the ACME_INT vector in the next free vector and allocate two paragrafs of data when its initialization routine gets called.

```
     MOV  BP, V_SYSTEM         ; HP vector 12H for EX-BIOS.
     MOV  AH, F_INS_FREEGETDS  ; Function 14H
     MOV  DI, CS               ; Get CS, IP of ACME_INT routines
     MOV  ES, DI
     MOV  DI, offset ACME_INT
     PUSH DS                   ; EX-BIOS Destroys DS
     CALL SYSCALL              ; Int for EX-BIOS (default 6FH)
     POP  DS
     MOV  VECTOR_NUMBER, BX    ; Save the vector number
                               ; routines are installed.
; ACME_INT routine handles initialization and
; allocates 2 paragraphs from EX-BIOS RAM for
; its data segment.

ACME_INT:
     CMP  AH, F_SYSTEM         ; Decode F_SYSTEM subfunction
     JNE  NOT_SUPPORTED        ; SF_INIT.
     CMP  AL, SF_INIT
     JE   ACME_INIT
NOT_SUPPORTED:                 ; Any unknown functions should
     MOV  AH, RS_UNSUPPORTED   ; return with RS_UNSUPPORTED
     IRET                      ; in AH.

ACME_INT:
     SUB  BX, 2                ; Decrement the "last used DS" passed
                               ; to us. This allocates 2 paragraphs
                               ; and makes our data segment the "last
                               ; used DS". Make sure to pass this new
                               ; BX back to F_INS_FREEGETDS code.
     MOV DS, BX                ; Now we can initialize the
                               ; data in our segment.
     ASSUME  DS:NOTHING
     MOV ACME_ATTR, 55AAH      ; Put data into Attribute word
     MOV ACME_NAME_INDEX, 55AAH ; Put a dummy index for now.
              .
              .                ; Initialize rest
              .                ; of data segment here.
     MOV AH, RS_SUCCESSFUL     ; Always return this status
                               ; if successful initialization.
     IRET

;
; Sample segment for this routine
;
     ACME_SEG struc
     ACME_ATTR dw 0            ; Attribute word of ACME's data segment.
     ACME_NAME_INDEX dw 0      ; Index name of ACME routine.
     ACME_REST db 28 dup (?)   ; rest of data segment
     ACME_SEG ends
```

F__INS__FREEGLBDS (AH = 16H)

Installs a vector in the next free entry of the HP__VECTOR__TABLE and calls it with an SF__INIT function. When F__INS__FREEGLBDS calls the initialization routine, it passes the data segment of the EX-BIOS global data area in the BX register.

WARNING

If the SF__INIT function returns with an error code of RS__FAIL (0FEH), the power-on self test sequence will be executed.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_INS_FREEGLBDS (16H)
       ES:DI = CS:IP of the routine

On Exit: AH = Return Status Code
              0 = RS_SUCCESSFUL

Registers Altered: AH, BP, DS
```

F__INS__FIND (AH = 18H)

This function is used to search the HP__VECTOR__TABLE for drivers that have equal or similar values in a specified field of their data segment. Parameters passed to the function specify the location of the 16-bit field, the bits within the field to be compared (and__mask) and the pattern of bits the field is to be compared with. Given a starting vector address, the function searches the vector table for the next driver that matches the conditions specified and returns its vector address in SI.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_INS_FIND (18H)
          AL = 0 then respond on equality to pattern
                 ((field) .AND. (and_mask)) = pattern
               2 then respond on non_equal
                 ((field) .AND. (and_mask)) <> pattern
          BX = and_mask
          DX = pattern
          SI = vector address to start the search from.
          DI = field to be used in the function, this
               is the offset into an HP header.

On Exit: AH = Return status
              0    = RS_SUCCESSFUL
              OFEH = RS_FAIL--No match found
         SI = Vector address of the first entry that matched.

Registers Altered: AH, BP, DS, SI
```

Example: Find a vector that has the value X5AXH ("X" means allow these digits to take any value) in its attribute header (the first word of the driver's data segment)

```
     MOV  BP, V_SYSTEM       ; HP vector 12H
     MOV  AH, F_INS_FIND     ; Function 18H
     MOV  AL, 0              ; Return RS_SUCCESSFUL when the value is equal
     MOV  DI, 0              ; Look in the first word of driver's data segment
     MOV  DX, 05A0H          ; Look for value '5A' in the middle of the word.
     MOV  BX, 0FF0H          ; Mask off the don't care parts.
     MOV  SI, 0              ; Start looking from the first vector position.
     PUSH DS                 ; EX-BIOS destroys DS
     CALL SYSCALL            ; Int for EX-BIOS (default 6FH)
     POP  DS
     CMP  AH, RS_SUCCESSFUL  ; See if it found a match ?
     JNE  VECTOR_NOT_FOUND
VECTOR_FOUND:                ; Yes
     MOV  SAVED_VECTOR, SI
          .
          .
          .
VECTOR_NOT_FOUND: ; No
          .
          .
          .
```

F__RAM__GET (AH = 1EH)

This function gets the segment pointers of the EX-BIOS free RAM area. Two pointers are returned by this function call. The "last used DS" pointer marks the first paragraph of EX-BIOS RAM that is free for use. The "max DS" pointer marks the lowest value that "last used DS" can have. Figure 8-1 shows how the EX-BIOS memory is organized.

See the F__RAM__RET memory function.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_RAM_GET (1EH)

On Exit: AH = RS_SUCCESSFUL
         BX = "last used DS"
         DX = "max DS"

Registers Altered: AH, BP, DS, BX, DX
```

F_RAM_RET (AH = 20H)

Sets the "last used DS" and "max DS" EX-BIOS pointers to the values passed in the BX and DX registers. This allows the calling routine to reserve a piece of the EX-BIOS memory.

CAUTION

The F_INS_FIXGETDS and F_INS_FREEGETDS functions described above also modify these values. Use caution when allocating memory with both methods.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_RAM_GET (20H)
          BX = "last used DS"
          DX = "max DS"

On Exit: AH = RS_SUCCESSFUL

Registers Altered: AH, BP, DS
```

Example: The following code allocates five paragraphs (80 bytes) of EX-BIOS memory.

```
;  Check Get the memory pointers first.
;
     MOV BP, V_SYSTEM    ; HP vector 12H.
     MOV AH, F_RAM_GET   ; function 1EH
     PUSH DS             ; EX-BIOS Destroys DS
     CALL SYSCALL        ; Int for EX-BIOS (default 6FH)
     POP DS
;
; Check to see if there is enough memory to allocate 5 paragraphs.
;
     SUB BX, 0005H       ; Create a new "last used DS" by
                         ; moving pointer towards "max DS".
     CMP BX, DX          ; Is "last used DS" >= "max DS"?
     JL NO_MEMORY_LEFT

ENOUGH_MEMORY_LEFT:      ; Yes: Allocate 5 paragraphs.
     MOV BP, V_SYSTEM    ; HP vector 12H
     MOV AH, F_RAM_RET   ; function 20H
     PUSH DS             ; EX-BIOS Destroys DS
     CALL SYSCALL        ; Int for EX-BIOS (default 6FH)
     POP DS
     MOV MEMORY_SEG, BX  ; Save this new memory pointer for later use
          .
          .              ; Continue
          .
```

```
NO_MEMORY_LEFT:          ; No:
;
; Typical thing to do here is to allocate more
; memory for the the EX-BIOS RAM and reboot
; system.
;
```

F_CMOS_GET (AH = 22H)

Read a byte from CMOS. It verifies the checksum on the industry standard CMOS area and returns RS_FAIL if the checksum is invalid.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_CMOS_GET (22H)
          BL = address of CMOS byte to read

On Exit: AH = Return Status Code
         AL = byte of data from CMOS

Registers Altered: AX, BP, DS.
```

F_CMOS_RET (AH = 24H)

Write a byte to CMOS. Calculate a new checksum for both the industry standard CMOS area and the HP CMOS area.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_CMOS_RET (24H)
          AL = byte of data to be written to CMOS
          BL = address of byte to be written to CMOS

On Exit: AH = Return Status Code

Registers Altered: AX, BP, DS.
```

Example: Make the monochrome display the primary video adapter by setting this information in the equipment byte of CMOS memory.

```
; Read the equipment byte.
;
     MOV  BP, V_SYSTEM   ; HP vector 12H.
     MOV  AH, F_CMOS_GET ; function 22H
     MOV  BL, 14H        ; Address of the equipment byte
     PUSH DS             ; EX-BIOS destroys DS
     CALL SYSCALL       ; Int for EX-BIOS (default 6FH)
     POP  DS
     CMP  AH, RS_FAIL    ; See if CMOS is valid
     JE   INVALID_CMOS
;
; Isolate the video and set appropiate video bits.
;
     AND  AL, 11001111B
     OR   AL, 00110000B  ; Select monochrome display

; Write the equipment byte.
;
     MOV  BP, V_SYSTEM   ; HP vector 12H
     MOV  AH, F_CMOS_RET ; function 24H
     PUSH DS             ; EX-BIOS destroys DS
     CALL SYSCALL       ; Int for EX-BIOS (default 6FH)
     POP  DS
     .
     .
     .
     INVALID_CMOS:
     .
     .
     .
```

F_YIELD (AH = 2AH)

Currently loads AH with RS_SUCCESSFUL and does an IRET. This is a "hook" for multi-tasking systems.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_YIELD (2AH)

On Exit: AH = Return Status Code

Registers Altered: AH, BP, DS
```

F__SND__CLICK__ENABLE (AH = 30H)

Enables the keyclick function.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_SND_CLICK_ENABLE (30H)

On Exit: AH = Return Status Code

Registers Altered: AH, BP, DS.
```

F__SND__CLICK__DISABLE (AH = 32H)

Disables the keyclick function, sets the EX-BIOS global data area T__SND__CLICK__DURA byte to zero.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_SND_CLICK_DISABLE (32H)

On Exit: AH = Return Status Code

Registers Altered: AH, BP, DS
```

F__SND__CLICK (AH = 34H)

This functions issues a keyclick.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_SND_CLICK (34H)

On Exit: AH = Return Status Code

Registers Altered: AH, BP, DS
```

F__SND__BEEP__ENABLE (AH = 36H)

Enables the beep function.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_SND_BEEP_ENABLE (36H)

On Exit: AH = Return Status Code

Registers Altered: AH, BP, DS
```

F__SND__BEEP__DISABLE (AH = 38H)

Disables the beep function.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_SND_BEEP_DISABLE (38H)

On Exit: AH = Return Status Code

Registers Altered: AH, BP, DS
```

F_SND_BEEP (AH = 3AH)

Makes a sound as defined by the current values of T_SND_BEEP_CYCLE and T_SND_BEEP_DURA in the EX-BIOS data area.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_SND_BEEP (3AH)

On Exit: AH = Return Status Code

Registers Altered: AH, BP, DS
```

F_SND_SET_BEEP (AH = 3CH)

Defines beep frequency and duration.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_SND_SET_BEEP (3CH)
          BX = Frequency 1 to 25000 hz.
               If (BX) = 0 then tone off.
          DX = duration of tone in 10 microsecond increments

On Exit: AH = Return Status Code

Registers Altered: AH, DS, BP.
```

Example: Set beep frequency to 660 Hz for duration of 1/2 second.

```
     MOV BP, V_SYSTEM          ; HP vector 12H
     MOV AH, F_SND_SET_BEEP    ; function 3CH
     MOV BX, 660               ; Frequency in hertz
     MOV DX, 50000             ; 1/2 a second in 10 microsecond
                               ; increments.
     PUSH  AH, F_SND_SET_BEEP
     PUSH DS                   ; EX-BIOS destroys DS
     CALL SYSCALL              ; Int for EX-BIOS (default 6FH)
     POP DS
```

F_SND_TONE (AH = 3EH)

Generates a tone of the given frequency and duration with an approximate 0.5 percent error.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_SND_TONE (3EH)
          BX = Frequency 1 to 25000hz. If (BX) = 0 then tone off.
          DX = Duration of tone in 10 microsecond increments.

On Exit: AH = Return Status Code

Registers Altered: AH, DS, BP
```

F_STR_GET_FREE_INDEX (AH = 40H)

Returns to caller the next string index that does not conflict with the ROM-based string indices.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_STR_GET_FREE_INDEX (40H)

On Exit: AH = RS_SUCCESSFUL
         BX = Next free index.

Registers Altered: AH, BX, DS, BP
```

Example: This example gets the next string index available to the user.

```
MOV BP, V_SYSTEM                ; HP vector 12H
MOV AH, F_STR_GET_FREE_INDEX    ; funct. 40H
PUSH DS                         ; EX-BIOS destroys DS
PUSH  BP, F_STR_GET_FREE_INDEX
CALL SYSCALL                    ; Int for EX-BIOS (default 6FH)
POP DS
MOV FIRST_FREE_INDEX,BX         ; Save it for later use.
.
.
.
```

F_STR_DEL_BUCKET (AH = 42H)

Finds a header with the given address and deletes it from the bucket header list.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_STR_DEL_BUCKET (42H)
          DI = offset address of bucket header
          ES = segment address of bucket header

On Exit: AH = RS_SUCCESSFUL if header found and deleted
              RS_FAIL if header not found.

Registers Altered: AH, DS, BP.
```

F_STR_PUT_BUCKET (AH = 44H)

Takes a header and its corresponding pointers and adds them to the front of the list.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_STR_PUT_BUCKET (44H)
          DI = Offset address of header
          ES = Segment address of header

On Exit: AH = RS_SUCCESSFUL

Registers Altered: AH, BP, DS.
```

Example: Adds a set of strings and its associated data structures for the ACME_INT driver.

```
; String data structures (see Figure 8-2)
;
     STR_HEADER STRUC
     STR_NXT_HDR DD (?)
     STR_UPPER_BOUND DW (?)
     STR_LOWER_BOUND DW (?)
     STR_LIST_PTR DD (?)
     STR_SEGMENT DW (?)
     STR_HEADER ENDS
;
; Now build a bucket (set of strings) for the
; ACME_INT:
;
; First list ACME_INT's strings:
     size_acme_name db l_acme_name -
     f_acme_name    =  $
     acme_name      db 'Acme Co.',0H
     l_acme_name    =  $
     size_item_1    db l_item_1 - f_item_1 - 1
     f_item_1       =  $
     item_1         db 'Hello World',0H
     l_item_1       =  $
     size_item_2    db l_item_2 - f_item_2 - 1
     f_item_2       =  $
     item_2         db 'Widgets',0H
     l_item_2       =  $
;
; Now build table of bucket pointers:
;
     acme_ptrs label near
     dw offset acme_name
     dw offset item_1
     dw offset item_2
```

```
;
; Now build the bucket header data structure
;
    acme_bucket label near
    dw 0FFFFH                   ; This is the only bucket.
    dw 0FFFFH
    dw 1002H                    ; Adding string indexes 1000..1002
    dw 1000H
    dw offset acme_ptrs         ; address of pointer list
    dw segment acme_ptrs
    dw segment acme_name        ; segment of all strings
;
; Do the function call to add bucket.
;
    MOV BP, V_SYSTEM            ; HP vector 12H
    MOV AH, F_STR_PUT_BUCKET    ; function 44H
    MOV DI, offset acme_bucket
    MOV ES, segment acme_bucket
    PUSH DS                     ; EX-BIOS Destroys DS
    CALL SYSCALL                ; Int for EX-BIOS (default 6FH)
    POP DS
```

F_STR_GET_STRING (AH = 46H)

Given an index, this function searches the list of bucket headers for the bucket pointer with the given index. It returns a pointer to the string.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_STR_GET_STRING (46H)
          BX = String index

On Exit: AH = RS_SUCCESSFUL if index found in a bucket
         CX = How many characters are in the string exclusive
              of the byte count and the zero byte at the end.
      DS:SI = Address of header where string was found.
      ES:DI = Pointer to first character of the string.

Registers Altered: AH, CX, SI, DI, BP, DS, ES
```

Example: Search for the name of the ACME_INT routine as index 1000H.

```
     MOV BP, V_SYSTEM           ; HP vector 12H
     MOV AH, F_STR_GET_STRING ; Function 46H
     MOV BX, 1000H              ; Index of ACME_INT name string
     PUSH DS                    ; EX-BIOS destroys DS
     CALL SYSCALL               ; Int for EX-BIOS (default 6FH)
;
; Write the string to the screen:
;
     MOV AX, F10_WRS_00         ; Call the write string function.
     MOV  BP, SI                ; Offset of string address
     PUSH DS                    ; Segment of string address
     POP  ES                    ; CX is already set
     MOV  DX, 0                 ; Cursor position at (0,0)
     MOV  BH, 0                 ; Video page 0
     MOV  BL, 7                 ; Character attribute
     INT  INT_VIDEO             ; Video interrupt 10
     POP  DS                    ; Recover old DS
```

F_STR_GET_INDEX (AH = 48H)

Given a pointer to a string, it returns the index of the string if it is in the bucket header list.

```
On Entry: BP = V_SYSTEM (12H)
          AH = F_STR_GET_INDEX (48H)
       ES:DI = Pointer to first character of
               the zero terminated string.

On Exit: AH = RS_SUCCESSFUL if index was found.
         BX = Index found for the given string.

Registers Altered: AH, BX, BP, DS
```

Example: Get the index of the ACME_NAME string.

```
     MOV  BP, V_SYSTEM         ; HP vector 12H
     MOV  AH, F_STR_GET_INDEX  ; function 48H
     MOV  DI, seg ACME_NAME    ; Move segment of string
     MOV  ES, DI               ; into ES
     MOV  DI, offset ACME_NAME
     PUSH DS                   ; EX-BIOS destroys DS
     PUSH BP,OFFSET ACME_NAME
     CALL SYSCALL             ; Int for EX-BIOS (default 6FH)
     POP  DS
     MOV  ACME_NAME_INDEX, BX  ; Save the index.
```

9

System Processes

This chapter describes system processes contained in the ROM BIOS. System processes are different from drivers in that they are not readily accessible to application programs and they perform larger tasks than a typical driver function. The ROM BIOS has five main system processes: reset, power-on self test (POST), system generation (SYSGEN), booting (BOOT), and return from protected mode.

Reset

The CPU is reset through a hardware reset signal. This signal sets the CS and IP registers to begin execution at memory location 0F000:0FFF0H. The system can be reset by either a hardware reset to the CPU, or by any software routine that jumps to memory location 0F000:0FFF0H. There are three events that initiate a system reset:

- **Power-on.** - This reset occurs when power is applied to the system. The power supply resets the CPU through its reset signal when the system is turned on. POST is initiated and performs a full memory test.

- **Soft Reset.** - This reset is initiated by the <Ctrl>-<Alt>-<DEL> key sequence. This sequence is interpreted by the INT 09H keyboard interrupt service routine as a reset command. POST is initiated. A full memory test is not performed.

- **Programmatic Reset.** - The final reset source is a software initiated hardware reset. A command is sent to the 8042 controller to pulse the CPU hardware reset line. Once the CPU has been placed in the Protected Mode, a hardware reset is the only method available to return to the Real Mode (the 80386 can return to Real Mode by using a MOV CR0 instruction). POST may or may not be performed depending upon the shutdown status byte in CMOS.

Once a reset operation has been initiated by one of the three possible sources, the system must determine if it is a power-on reset. If it is a power-on reset, bit 2 in the 8042 controller's status port is cleared. POST is performed. A command is sent to the 8042 to set bit 2. If it is not a power-on reset, bit 2 in the 8042 controller status port is already set. The CMOS shutdown status byte determines whether POST is performed.

If it is not a power-on reset, the system looks at the shutdown status byte (CMOS address 0FH) to determine whether to perform POST or return from protected mode. If the shutdown status byte is set to one of the values that indicates the system is returning from protected mode, the reset process will initiate the return from protected mode process. This process is described next. All other values of the shutdown status byte are interpreted as reset commands, and the reset process will initiate the power-on self test process. The reset process has completed its tasks when one of these two processes has been invoked.

Protected Mode Support

The CPU has two modes of operation: **Real mode** and **Protected mode**. **Real mode** provides a 1 MB address space and is 8086 compatible. **Protected mode** provides memory protection, virtual memory addressing, and either a 16 MB (for the 80286 CPU), or a 4 gigabyte (for the 80386 CPU) physical address space. The normal mode of operation of the system is real mode. However, a few programs use protected mode, for example, VDISK.SYS, the MS-DOS virtual disc device driver.

Additionally, the 80386 provides a third mode which is a subset of Protected mode: **Virtual 8086 mode.** In **Virtual 8086 mode**, an application would run as it would on an 8086 machine. This mode allows multi-tasking with older MS-DOS applications; each application allotted up to 1 MB of memory address space.

The system provides some support to the programmer for use of the protected mode features. The INT 15H driver provides two functions that support system operation in protected mode. One of these functions enables data to be moved to and from extended memory. This function enters protected mode to perform this task, and returns to real mode. The second function provides a method for programmers to switch into protected mode. These functions are described in Chapter 8 of this manual.

Shutdown Status Byte

The shutdown status byte is used by the system to determine what action should be taken on reset. Table 9-1 shows how the shutdown status byte is interpreted. Note that any value that does not indicate a return from protected mode is interpreted by the system as a reset, and will cause the reset process to invoke POST.

Table 9-1. Shutdown Status Byte

Value	Definition
00-04H	Perform power-on reset sequence.
05H	Flush keyboard and jump via double word stored at 0040:0067H.
06-07H	Perform power-on reset sequence.
08H	Return from test of extended memory.
09H	Return from INT 15H block move function.
0AH	Jump via double word stored at 0040:0067H.
0BH-FFH	Perform power-on reset sequence.

The values 08H and 09H are used internally by the ROM BIOS. If the return from protected mode process detects either of these values, it will branch to their respective routines. Values 05H and 0AH should be used by all other programs returning from protected mode.

Power-On Self Test (POST)

Each time the system is powered on, or a reset is performed, the POST process is executed. The purpose of the POST process is to verify the basic functionality of the system components and to initialize certain system parameters. The POST process performs the following tasks:

- Test the operation of the CPU.
- Test the system ROM.
- Test and initialize 8254 timer/counter and start the refresh counter.
- Test the first 64 KB of system RAM.
- Test memory cache subsystem (Vectra RS/20C and RS/25C only.)
- Initialize the video display for diagnostic messages.
- Test and initialize DMA controllers and DMA page registers.
- Test and initialize the 8259A interrupt controllers.
- Test the 8042 controller and Scandoor.
- Test the HP-HIL controller.
- Test CMOS RAM for integrity.
- Determine if manufacturing electronic tool is present. If so, run manufacturing test.
- Test the remaining base system RAM (RAM above the first 64 KB).
- Test the extended RAM above memory address 100000H (protected mode RAM.)
- Test the real-time clock portion of the RTC/CMOS chip.
- Test the keyboard interface and the keyboard itself.
- Test the flexible disc controller subsystem.
- Test the coprocessor if present (80287 for Vectra ES series, 80387 for Vectra QS series, and 80387 and Weitek coprocessor for Vectra RS series).
- Test the CPU clock speed.
- Test serial port.

The power-on self test performs tests on various subsystems in the hardware when power is switched on or when the system is reset. If a problem is detected, a 4 digit hex error code is displayed and four short beeps are sounded. (In order for all codes to be displayed, the video display adapter must be a multimode, a monochrome, or a color adaptor.) These codes are listed in Tables 9-2a (for **Vectra ES series**) and 9-2b (for **Vectra QS and RS series**).

Table 9-2a and 9-2b Legend:

Error Code is in the form of 4 hex digits; X, Y, Z are hex digits.
x = don't care bit in the hex digit
b = valid bit in the hex digit
CGA = Clock Gate Array chip
kbd = keyboard
MFG = Manufacturing

Table 9-2a. Vectra ES POST Error Code Listing

Code	Test	Chip	Description
000F	80286	U810	80286 CPU is bad
0010	ROM	U28	Bad checksum on ROM 0
0011	ROM	U27	Bad checksum on ROM 1.
011X	RTC	U108	One of the RTC (Real Time Clock) registers is bad. Reg # = X (0 - D).
0120	RTC		RTC failed to tick.
0240	CMOS	U108	CMOS/RTC has lost power.
0241	CMOS		Invalid checksum on IBM CMOS area.
0280	CMOS		Invalid checksum on HP CMOS area.
02XY	CMOS		One of the CMOS registers is bad Reg # = XY - 40 Example: 024E = reg #E is bad
0301	8042	U1010	8042 failed to accept the RESET cmd.
0302	8042		8042 failed to respond to the reset cmd.
0303	8042		8042 failed on RESET.
0311	8042		8042 failed to accept the "WRITE CMD BYTE" cmd.
0312	8042		8042 failed to accept the data of the above cmd.
0321	Scandoor	U128	8042 failed to accept scancode from port 68.
0322	Scandoor		8042 failed to respond to the above scancode. (This will happen when keyboard is locked up.)
0323	Scandoor		8042 responded incorrectly to the above scancode.
0331	Scandoor		8042 failed to accept cmd (command) from port 6A.
0332	Scandoor		8042 failed to generate SVC on port 67.
0333	Scandoor		8042 generate incorrect HPINT type on port 65.
0334	Scandoor		8042 failed the r/w register test on port 69.
0335	Scandoor		8042 failed to generate a HPINT on IRQ 15
0336	Scandoor		8042 failed to generate a HPINT on IRQ 12
0337	Scandoor		8042 failed to generate a HPINT on IRQ 11
0338	Scandoor		8042 failed to generate a HPINT on IRQ 10
0339	Scandoor		8042 failed to generate a HPINT on IRQ 7
033A	Scandoor		8042 failed to generate a HPINT on IRQ 5
033B	Scandoor		8042 failed to generate a HPINT on IRQ 4
033C	Scandoor		8042 failed to generate a HPINT on IRQ 3

Table 9-2a. Vectra ES POST Error Code Listing (Cont.)

Code	Test	Chip	Description
0341	Keyboard		8042 failed to accept the kbd interface test cmd.
0342	Keyboard		8042 failed to repsond to the kbd interface test cmd.
0343	Keyboard		Kbd interface test failed: kbd clock line stuck low.
0344	Keyboard		Kbd interface test failed: kbd clock line stuck high.
0345	Keyboard		Kbd interface test failed: kbd data line stuck low.
0346	Keyboard		Kbd interface test failed: kbd data line stuck high.
0350	Keyboard		No acknowledgement from kbd self test cmd.
0351	Keyboard		Bad acknowledgement from kbd self test cmd.
0352	Keyboard		Kbd is dead or not connected.
0353	Keyboard		No result from kbd self test cmd.
0354	Keyboard		Kbd self test failed.
06XX	Keyboard		Kbd has stuck key: XX = scancode of stuck key.
0401	8042	U1010	8042 failed to enable Gate A20.
0503	Serial Port		Serial Port dead or non-existent.
0505	Serial Port		Serial Port fails register tests
0543	Parallel Port		Parallel Port dead or non-existent.
0700	CGA	U410	Failed to switch to SLOW mode
0701	CGA		Failed to switch to DYNAMIC mode
0702	CGA		Timer (channel 0) failed to interrupt
0703	CGA		Memory cycles too slow in SLOW mode
0704	CGA		Memory cycles too fast in SLOW mode
0705	CGA		IO cycles too slow in SLOW mode
0706	CGA		IO cycles too fast in SLOW mode
0707	CGA		Memory cycles too slow in DYNAMIC mode
0708	CGA		Memory cycles too fast in DYNAMIC mode
0709	CGA		IO cycles too slow in DYNAMIC mode
070A	CGA		IO cycles too fast in DYNAMIC mode
110X	Timer	U108	One of the timer channels failed register test. X (0 - 2) = timer channel that failed the test.
1200	Timer		Memory Refresh signal stuck high.
1201	Timer		Memory Refresh signal stuck low.
211X	DMA	U108	DMA #1 failed on register r/w (read/write) test. Reg # = X (0 - 7).
212X	DMA		DMA #2 failed on register r/w test. Reg # = X (0 - 7).
221X	DMA		DMA page registers bad X (0 - 7) = bad register

Table 9-2a. Vectra ES POST Error Code Listing (Cont.)

Code	Test	Chip	Description
300X	HP-HIL Controller	U1210	HP-HIL Controller chip failed the self test. X = xxx1 => r/w fail with data = 0da5h X = xx1x => r/w fail with data = 0d5ah X = x1xx => r/w fail with data = 0aa5h X = 1xxx => r/w fail with data = 0a5ah Note: the above may be or'ed together to generate more complex error codes.
3010	HP-HIL Device		HP-HIL device test failed.
4XYZ	RAM R/W		RAM in lower 640K failed the R/W test. X = bbbx => bbb (0-7) is # of 128K bank bbb0 => indicate even byte bad bbb1 => indicate odd byte bad YZ = bbbb bbbb => bits for which b=1 are bad. Follow the procedure below to identify the bad RAM chip(s) on the processor PCA. For X = 0, 2, 4, or 6, interpret YZ as follow: Y <> 0 => U23 is bad Z <> 0 => U13 is bad For X = 1, 3, 5, or 7, interpret YZ as follow: Y <> 0 => U43 is bad Z <> 0 => U33 is bad For X = 8, interpret YZ as follow: Y <> 0 => U22 is bad Z <> 0 => U12 is bad For X = 9, interpret YZ as follow: Y <> 0 => U42 is bad Z <> 0 => U32 is bad
5XYZ	RAM Marching Ones		RAM in lower 640K failed the marching one test. X = bbbx => bbb (0-7) is # of 128K bank bbb0 => indicate even byte bad bbb1 => indicate odd byte bad YZ = bbbb bbbb => bits for which b=1 are bad. Use the same procedure outlined for the 4XYZ error code to identify bad RAM chip(s) on the processor PCA for the marching ones test.
61XY	RAM addr Indepen-dence		Some address lines to RAM are stuck to 0 or 1. XY = 00bb bbbb => RAM address line bbbbbb is stuck. XY = 01bb bbbb => Multiple address lines are stuck. bbbbbb is the first bad one.

Table 9-2a. Vectra ES POST Error Code Listing (Cont.)

Code	Test	Chip	Description
620X	RAM Parity		Parity error has occurred during RAM tests on the lower 640K of RAM. X = address in 64K bank where parity occurred. If X = 0 to 7 U21 or/and U31 is/are bad. If X = 8 to 9, U11 or/and U41 is/are bad.
63XY	IO Channel Check		Parity error has occurred during RAM tests above the 1st MB (i.e., extended RAM on the I/O channel). XY = address in 64K bank where parity occurred.
6400	Parity Ckt	U97	The parity generator circuit failed to generate parity error when unitialized RAM was read at power up.
71XY	Master 8259 Mask	U108	Master 8259 failed the r/w test on its mask register. XY = bbbb bbbb => bits in which b=1 is bad.
72XY	Slave 8259 Mask	U108	Slave 8259 failed the r/w test on its mask register. XY = bbbb bbbb => bits in which b=1 is bad.
7400	Master 8259 Interrupt	U108	Master 8259 failed the interrupt test. Note that this test uses the interval timer channel 0 to generate the interrupt.
7500	Slave 8259 Interrupt	U108	Slave 8259 failed the interrupt test. Note that this test uses the RTC to generate the interrupt.
9XYZ	Flexible Disc Subsystem		Error in Flexible Disc Controller (FDC) test. In POST, flexible disc error is one word, the primary report format. In Strife/MFG, the error is two word, primary and secondary report:

Table 9-2a. Vectra ES POST Error Code Listing (Cont.)

Code	Test	Chip	Description
Primary Report Format: 9XYZ			X = flexible drive # (i.e. 0 = A:, 1 = B:) Y = 0 indicates 1st level error For 1st level error, Z = 0 = unsuccessful input from FDC 1 = unsuccessful output to FDC 2 = error while executing a seek 3 = error while executing a recalibrate 4 = error while verifying ram buffer 5 = error while resetting FDC 6 = wrong drive identified 7 = wrong media identified 8 = no interrupt from FDC 9 = failed to detect track 0 A = failed to detect index pulse Y > 0 indicates higher level error 1 = read sector error side 0 2 = read sector error side 0 3 = write sector error side 1 4 = write sector error side 0 5 = format sector error side 0 6 = format sector error side 1 7 = read ID error side 0 8 = read ID error side 1 For higher level errors, Z = 1 = no ID address mark 2 = no data address mark 3 = media is write protected 4 = sector number wrong 5 = cylinder number wrong 6 = bad cylinder 7 = DMA overrun 8 = ID CRC error 9 = Data CRC error A = End of cylinder B = Unrecognizable error
Secondary Report Format: 9XYZ			XY = xbbb bbbb where bbb bbbb is the cylinder number where failure occurred. Z = sector # where failure occurred.

Table 9-2a. Vectra ES POST Error Code Listing (Cont.)

Code	Test	Chip	Description
A001	80287	U210	No 80287 is detected. This error code will not be reported in POST.
A002	80287		80287 failed the R/W test on its stack registers.
A00C	80287		80287 failed to generate an zero-divide interrupt.
CXYZ	Extended RAM		R/W test failure on extended RAM. X = 0 => even byte is bad. X = 1 => odd byte is bad. YZ = address in 64K bank where RAM failed. Since there could be many different type of RAM chips used in the extended memory, we will not provide the method here to identify the bad RAM chip(s) on the extended memory board.
CFFF	Extended RAM		No extended RAM is found. This error code will not be reported in POST.
EXYZ	Extended RAM		Marching one test failure on extended RAM. X = 0 => even byte is bad. X = 1 => odd byte is bad. YZ = addr in 64K bank where RAM failed. Since there could be many different type of RAM chips used in the extended memory, we will not provide the method here to identify the bad RAM chip(s) on the extended memory board.

Table 9-2b. Vectra QS and RS POST Error Code Listing

Code	Test	Description
000F	80386	80386 CPU is bad.
0010	ROM	Bad checksum on ROM 0.
0011	ROM	Bad checksum on ROM 1.
011X	RTC	One of the RTC (Real Time Clock) registers is bad. Reg # = X (0 - D).
0120	RTC	RTC failed to tick.
0240	CMOS	CMOS/RTC has lost power.
0241	CMOS	Invalid checksum on IBM CMOS area.
0280	CMOS	Invalid checksum on HP CMOS area.
02XY	CMOS	One of the CMOS registers is bad Reg # = XY - 40 Example: 024E = reg #E is bad
0301	8042	8042 failed to accept the RESET cmd.
0302	8042	8042 failed to respond to the RESET cmd.
0303	8042	8042 failed on RESET.
0311	8042	8042 failed to accept the "WRITE CMD BYTE" cmd.
0312	8042	8042 failed to accept the data of the above cmd.
0321	Scandoor	8042 failed to accept scancode from port 68.
0322	Scandoor	8042 failed to respond to the above scancode. (This will happen when keyboard is locked up.)
0323	Scandoor	8042 responded incorrectly to the above scancode.
0331	Scandoor	8042 failed to accept cmd (command) from port 6A.
0332	Scandoor	8042 failed to generate SVC on port 67.
0333	Scandoor	8042 generate incorrect HPINT type on port 65.
0334	Scandoor	8042 failed the r/w register test on port 69.
0335	Scandoor	8042 failed to generate a HPINT on IRQ 15
0336	Scandoor	8042 failed to generate a HPINT on IRQ 12
0337	Scandoor	8042 failed to generate a HPINT on IRQ 11
0338	Scandoor	8042 failed to generate a HPINT on IRQ 10
0339	Scandoor	8042 failed to generate a HPINT on IRQ 7
033A	Scandoor	8042 failed to generate a HPINT on IRQ 5
033B	Scandoor	8042 failed to generate a HPINT on IRQ 4
033C	Scandoor	8042 failed to generate a HPINT on IRQ 3
0341	Keyboard	8042 failed to accept the kbd interface test cmd.
0342	Keyboard	8042 failed to repsond to the kbd interface test cmd.
0343	Keyboard	Kbd interface test failed: kbd clock line stuck low.
0344	Keyboard	Kbd interface test failed: kbd clock line stuck high.
0345	Keyboard	Kbd interface test failed: kbd data line stuck low.
0346	Keyboard	Kbd interface test failed: kbd data line stuck high.
0350	Keyboard	No acknowledgement from kbd self test cmd.
0351	Keyboard	Bad acknowledgement from kbd self test cmd.

Table 9-2b. Vectra QS and RS POST Error Code Listing (Cont.)

Code	Test	Description
0352	Keyboard	Kbd is dead or not connected.
0353	Keyboard	No result from kbd self test cmd.
0354	Keyboard	Kbd self test failed.
06XX	Keyboard	Kbd has stuck key: XX = scancode of stuck key.
0401	8042	8042 failed to enable Gate A20.
0503	Serial Port	Serial Port dead or non-existent.
0505	Serial Port	Serial Port fails register tests.
	Clock Speed Test for:	
0700	82C301	Failed to switch to SLOW speed
0701	82C301	Failed to switch to FAST speed
0702	82C206	Timer failed to interrupt
0703	82C301	CPU clock too slow in SLOW speed
0704	82C301	CPU clock too fast in SLOW speed
0707	82C301	CPU clock too slow in FAST speed
0708	82C301	CPU clock too fast in FAST speed
0709	82C301	Failed to switch to ATCLK for BUS clock
070B	82C301	CPU clock too slow at MEDIUM speed.
070C	82C301	CPU clock too fast at MEDIUM speed.
110X	Timer	One of the timer channels failed register test. X (0 - 2) = timer channel that failed the test.
1200	Timer	Memory Refresh signal stuck high.
1201	Timer	Memory Refresh signal stuck low.
211X	DMA	DMA #1 failed on register r/w (read/write) test. Reg # = X (0 - 7).
212X	DMA	DMA #2 failed on register r/w test. Reg # = X (0 - 7).
221X	DMA	DMA page registers bad X (0 - 7) = bad register
300X	HP-HIL Controller	HP-HIL Controller chip failed the self test. X = xxx1 => r/w fail with data = 0da5h X = xx1x => r/w fail with data = 0d5ah X = x1xx => r/w fail with data = 0aa5h X = 1xxx => r/w fail with data = 0a5ah Note: the above may be or'ed together to generate more complex error codes.

Table 9-2b. Vectra QS and RS POST Error Code Listing (Cont.)

Code	Test	Description
3010	HP-HIL Device	HP-HIL device test failed.
4XYZ	RAM R/W	RAM in lower 640K failed the R/W test. X = bbcc => bb is # of 64K of 32-bit word bank cc = 00 => byte 0 is bad 01 => byte 1 is bad 10 => byte 2 is bad 11 => byte 3 is bad YZ = bbbb bbbb => bits for which b=1 are bad.
5XYZ	RAM Marching Ones	RAM in lower 640K failed the marching one test. X = bbcc => bb is # of 64K of 32-bit word bank cc = 00 => byte 0 is bad 01 => byte 1 is bad 10 => byte 2 is bad 11 => byte 3 is bad YZ = bbbb bbbb => bits for which b=1 are bad.
61XY	RAM addr Indepen- dence	Some address lines to RAM are stuck to 0 or 1. XY = 00bb bbbb => RAM address line bbbbbb is stuck. XY = 01bb bbbb => Multiple address lines are stuck. bbbbbb is the first bad one.
620X	RAM Parity	Parity error has occurred during RAM tests on the lower 640K of RAM. X = address in 64K bank where parity occurred.
63XY	IO Channel Check	Parity error from memory installed in the I/O channel during the above RAM tests. XY = address in 64K bank where parity occurred.
6500	Shadow RAM	Shadow RAM is bad at BIOS segment.
6510	Shadow RAM	Shadow RAM is bad at HP EGA segment.
71XY	Master 8259 Mask	Master 8259 failed the r/w test on its mask register. XY = bbbb bbbb => bits in which b=1 is bad.
72XY	Slave 8259 Mask	Slave 8259 failed the r/w test on its mask register. XY = bbbb bbbb => bits in which b=1 is bad.

Table 9-2b. Vectra QS and RS POST Error Code Listing (Cont.)

Code	Test	Description
7400	Master 8259 Interrupt	Master 8259 failed the interrupt test. Note that this test uses the interval timer channel 0 to generate the interrupt.
7500	Slave 8259 Interrupt	Slave 8259 failed the interrupt test. Note that this test uses the RTC to generate the interrupt.
9XYZ	Flexible Disc Subsystem	Error in Flexible Disc Controller (FDC) test. In POST, flexible disc error is one word, the primary report format. In Strife/MFG, the error is two word, primary and secondary report:

Table 9-2b. Vectra QS and RS POST Error Code Listing (Cont.)

Code	Test	Description
Primary Report Format: 9XYZ		X = flexible drive # (i.e. 0 = A:, 1 = B:) Y = 0 indicates 1st level error For 1st level error, Z = 0 = unsuccessful input from FDC 1 = unsuccessful output to FDC 2 = error while executing a seek 3 = error while executing a recalibrate 4 = error while verifying ram buffer 5 = error while resetting FDC 6 = wrong drive identified 7 = wrong media identified 8 = no interrupt from FDC 9 = failed to detect track 0 A = failed to detect index pulse Y > 0 indicates higher level error 1 = read sector error side 0 2 = read sector error side 0 3 = write sector error side 1 4 = write sector error side 0 5 = format sector error side 0 6 = format sector error side 1 7 = read ID error side 0 8 = read ID error side 1 For higher level errors, Z = 1 = no ID address mark 2 = no data address mark 3 = media is write protected 4 = sector number wrong 5 = cylinder number wrong 6 = bad cylinder 7 = DMA overrun 8 = ID CRC error 9 = Data CRC error A = End of cylinder B = Unrecognizable error
Secondary Report Format: 9XYZ		XY = xbbb bbbb where bbb bbbb is the cylinder number where failure occurred. Z = sector # where failure occurred.

Table 9-2b. Vectra QS and RS POST Error Code Listing (Cont.)

Code	Test	Description
A001	80387	No 80387 detected. POST will not report this error code.
A002	80387	80387 failed the R/W test on its stack registers.
A00C	80387	80387 failed to generate an zero-divide interrupt.
AF00	Weitek	Weitek* coprocessor (COP) Test failed to enter Protected Mode. (* indicates for Vectra RS only.)
AF01	Weitek	Weitek* coprocessor not present (will not be reported in POST.)
AF02	Weitek	Weitek* coprocessor failed Registers Test.
AF05	Weitek	Weitek* coprocessor failed Addition Test.
AF06	Weitek	Weitek* coprocessor failed Multiplication Test.
AF0C	Weitek	Weitek* coprocessor failed Interrupt Test.
B300	8042 **	Failed to switch to protected mode. (**** indicates errors detected by Memory Cache Test.**)
B301-B307	82385	General cache subsystem failure.
B400-B7FF	Main Memory **	Read/write test of DRAM locations 60000h-6FFFFh failed. Decode bits in error code to isolate failing memory module: BXYZ where X = 01aa => aa specifies which byte is bad (0 -3) YZ = bbbb bbbb => b=1 specifies bad bit e.g.: 0100 0010 => bits 6 and 1 bad
B800-BBFF	Static RAM	Read/write test of SRAM failed. Decode bits in error code to isolate failing chips: BXYZ where X = 10aa => aa specifies which byte is bad (0 - 3) YZ = bbbb bbbb => b=1 specifies bad bit e.g.: 0100 0010 => bits 6 and 1 bad
BC00-BFFF	Static RAM	Marching ones test of SRAM failed. Decode bits in error code to isolate failing chips: BXYZ where X = 11aa => aa specifies which byte is bad (0 - 3) YZ = bbbb bbbb => b=1 specifies bad bit e.g.: 0100 0010 => bits 6 and 1 bad
CXYZ	Extended RAM	R/W test failure on extended RAM. X = 0 => even byte is bad. X = 1 => odd byte is bad. YZ = address in 64K bank where RAM failed. Since there could be many different types of RAM chips used in the extended memory, we will not provide the method here to identify the bad RAM chip(s) on the extended memory board.
CFFF	Extended RAM	No extended RAM found. POST will not report this error code.
EXYZ	Extended RAM	Marching one test failure on extended RAM. X = 0 => byte 0 is bad. 1 => byte 1 is bad. 2 => byte 2 is bad. 3 => byte 3 is bad. YZ = addr in 64K bank where RAM failed. Since there could be many different type of RAM chips used in the extended memory, we will not provide the method here to identify the bad RAM chip(s) on the extended memory board.

If the POST process is initiated by a soft reset, the RAM tests and the cache memory test are not executed. This portion of POST determines the amount of system memory and performs a test of that memory. In all other aspects, POST executes the same for power-on, hard reset, and soft reset.

SYSGEN then compares the configuration information stored in the CMOS memory with the actual system. If a discrepancy is found, a message will be displayed instructing the user to run the SETUP program. For example, if the CMOS memory indicates two flexible disc drives present, but the system contains only one, the message will be displayed.

System Generation (SYSGEN)

When the POST code module has completed its tasks, it initiates the system generation (SYSGEN) process. The SYSGEN process initializes the system software, then initiates the boot process. In general, the system data structures are initialized by the SYSGEN process, whereas the system hardware is initialized by the POST process. For example, the STD-BIOS and EX-BIOS data areas are initialized by the SYSGEN process. SYSGEN initializes the following items:

- Interrupt vectors
- STD-BIOS data area
- EX-BIOS data area

The interrupt vectors are initialized to their default values. Processor interrupt vectors are initialized to their appropriate service routines. Hardware interrupt vectors are initialized to their service routines, or a null routine if they are unused. The interrupt vectors used to access the STD-BIOS drivers are initialized to their respective driver entry points.

The STD-BIOS data area fields are initialized to their default values. Configuration dependent fields such as the base I/O address of the serial and parallel ports, current video mode, etc. are initialized at this time.

The EX-BIOS data area is set up next in the SYSGEN process. Initializing the EX-BIOS data area consists of several distinct steps as outlined below.

Memory Allocation

The first step in the process is to allocate system memory for the EX-BIOS data area. This memory allocation algorithm has two important features. First, by taking the memory size stored in CMOS memory into consideration, it allows large driver data areas to be allocated in the EX-BIOS data area. This method of expanding the EX-BIOS data area is explained in Chapter 8. Second, it prevents invalid CMOS memory size data from preventing the system from booting. If the CMOS memory size is set (using the SETUP program or writing directly to the CMOS memory) such that there is insufficient room for the EX-BIOS data area, this algorithm will adjust the value and write the new value to CMOS memory. The EX-BIOS data area is required to support the EX-BIOS extended features.

There are three important variables in this calculation.

- RAM_SIZE--This is the top of actual system memory. It is usually 640 KB (system memory can be reconfigured as 256 or 512) and will always be an even multiple of 64 KB.

- EX-BIOS_SIZE--This variable is the size of the EX-BIOS data area, which is 4 KB in its default configuration.

- CMOS_SIZE--This is the memory size stored in CMOS.

The CMOS_SIZE is checked for validity. If it is between 4 KB and 64 KB from RAM_SIZE, this value is used as the base of the EX-BIOS data area. If CMOS_SIZE is more than 64 KB from RAM_SIZE, the base of the EX-BIOS data area is located 64 KB below the top of actual system memory. Finally, if CMOS_SIZE is less than 4 KB from the top of RAM_SIZE (or greater than the top of actual memory), the base of the EX-BIOS data area is located 4 KB from the top of system memory. The following formulas show this relationship:

If (RAM_SIZE--CMOS_SIZE) >n 4 KB and < 64 KB,
then EX-BIOS_SIZE = (RAM_SIZE--CMOS_SIZE).

If (RAM_SIZE--CMOS_SIZE) >n 64 KB,
then EX-BIOS_SIZE = 64 KB.

If (RAM_SIZE--CMOS_SIZE) < 4 KB,
then EX-BIOS_SIZE = 4 KB.

The following examples illustrate this relationship:

In a 640 KB system, if CMOS_SIZE is 512 KB, then the EX-BIOS_SIZE data area starts at 576 KB. This leaves an 64 KB free area between the EX-BIOS_SIZE data area and the memory allocated to DOS.

In a 640 KB system, if CMOS_SIZE is 620 KB, then the EX-BIOS_SIZE data area starts at 620 KB. In this case the EX-BIOS_SIZE data area occupies all the area between the top of RAM and the memory allocated to DOS.

The HP_VECTOR_TABLE Initialization

Once the EX-BIOS data area has been allocated, and its base address determined, the HP_VECTOR_TABLE is constructed. An image of the default HP_VECTOR_TABLE is stored in the system ROM. This image is transferred from ROM to the base of the EX-BIOS data area. All free and reserved vectors are initialized to point at V_DOLITTLE, a null routine. Some of these vectors will be initialized to other drivers later in the SYSGEN process.

EX-BIOS Driver Initialization

The next step in the SYSGEN process is the initialization of the EX-BIOS drivers. Each driver is called with the SF_INIT subfunction. Some of the EX-BIOS drivers add vectors to the table when called to initialize. For example, the V_HPHIL driver initializes the vector addresses reserved for the HP-HIL physical device drivers. The HP_VECTOR_TABLE is fully initialized to its default state when each driver has been called in this manner. Additional drivers may be added or substituted by application programs or system software utilizing the vector maintenance functions of V_SYSTEM (refer to Chapter 8 for a description of these functions).

Adapter and Option ROM Module Integration

The ROM BIOS architecture allows code modules residing on adapter cards to be integrated into the system. These ROM modules must be in the system address range of 0C0000H - 0DFFFFH. (Note that only video adapter cards can have base address in the range of 0C0000H through 0C7FFFH). In addition to ROM modules located on adapter cards, the Processor PCA contains additional sockets for option ROMs. These option ROMs are addressed from 0E0000H - 0EFFFFH. ROM modules located on adapter cards or on the Processor PCA are integrated into the system in the same manner.

All ROM modules contain a header and checksum byte. The header format is shown below:

Byte 0--55H
Byte 1--0AAH
Byte 2--Length of ROM module in 512 byte blocks.
Byte 3--Initialization entry point.

Bytes 0 and 1 are signature bytes. All ROM modules must contain this signature at the start of the header in order to be identified by the SYSGEN process.

Byte 2 of the header contains the number of 512 byte blocks in the ROM module, except the ROM module located on the Processor PCA (memory address 0E0000H). Byte 2 in that ROM module header is reserved.

During the boot process, the address range from 0C0000H to 0DFFFFH is scanned in 2 KB blocks looking for valid option ROM headers. In addition, memory location 0E0000H is also examined for a valid header. Since the scan does not proceed past 0E0000H, only one ROM module can reside in the address range 0E0000H to 0EFFFFH. The Processor PCA will accept two different size ROMs, 32 KB or 64 KB. If a 32 KB part is installed, the ROM will appear in the system address space starting at location 0E8000H instead of 0E0000H. Therefore, the 32 KB ROM will not be integrated into the system by SYSGEN.

If a valid ROM header is found, a checksum is computed for the ROM module. This is done by summing each byte in the ROM module. The sum of all the bytes in the ROM, including the checksum byte, must equal 0. For ROM modules located from 0C0000H to 0DFFFFH, the checksum is computed for the number of bytes indicated in the length field of the header. For a ROM module located from 0E0000H to 0EFFFFH, this checksum is calculated on the entire 64 KB of address space.

If the checksum is valid, a FAR call to byte 3 of the module is performed. The ROM module should perform any initialization required and then execute a RETF instruction.

This integration process allows option ROMs to install vectors in either the HP_VECTOR_TABLE or the low memory interrupt vectors. This re-vectoring process is the typical method used to integrate ROM modules into the system.

Shadow RAM (HP Vectra QS and RS Series Only)

On the **HP Vectra QS and RS series**, ROM integration is enhanced by a technique called **Shadow RAM** which speeds up system performance. **Shadow RAM** is a process where ROM is copied into high-speed 32-bit RAM addressed at the same physical location. This provides faster access to ROM-based video subsystems (such as HP's Enhanced Graphics Adapter) as well as HP Vectra QS and RS system BIOS firmware. This process is completed by the firmware during the power-up process and is completely transparent to applications.

Boot Process (INT 19H)

The boot process loads the operating system. The ROM BIOS INT 19H loads the boot sector from drive "A:" or "C:". This sector must contain the bootstrap loader for the operating system. Control is then passed to the code loaded from the boot sector. This code is responsible for loading the operating system. Refer to the appropriate operating system reference documentation for additional information on its boot process.

Booting From a Flexible Disc

The INT 19H driver attempts to read the boot sector from Drive "A:" (disc 0). It will retry the read four times before failing. The boot sector on flexible discs is located on Side 0, Track 0, Sector 1. Table 9-3 contains a description of the contents of a valid boot sector. If drive "A:" contains a disc that does not have a valid boot sector, then the system will report the error message:

Non-System disc or disc error
Replace and strike any key when ready.

If a valid boot sector is found, it is read into memory starting at location 07C0H:0000H (07C00H) and control is transferred through a FAR JUMP to location 07C0H:0000H. It is the responsibility of this code to load the rest of the operating system into memory.

Booting From a Hard Disc

If the flexible disc drive does not contain a disc, the system will attempt to boot from the hard disc. Booting from a hard disc is a two-step process. First, the active partition must be determined, then the boot record is read from the active partition.

The hard disc can be divided into as many as four partitions. Each partition contains an operating system, programs, and data. Only one of the partitions can be active at any time. Partitions are added, deleted, activated, and deactivated using utilities provided with the respective operating systems. Partitions occupy a specified number of cylinders on the disc. For example, let's say an optional 20 MB hard disc drive has

606 cylinders. One partition might occupy cylinders 0 through 303, while the second partition occupies cylinders 304 through 605. If the active partition does not contain an operating system, the system will report the error message indicating such.

Table 9-3. Boot Record

Offset	Size	Description
0000H	3 Bytes	Near JUMP instruction to boot code.
0003H	8 Bytes	OEM name and version number.
000BH	1 Word	Bytes per sector.
000DH	1 Byte	Sectors per allocation unit.
000EH	1 Word	Reserved sectors.
0011H	1 Byte	Number of File Allocation Tables (FATs).
0012H	1 Word	Number of root directory entries.
0014H	1 Word	Number of sectors in logical image.
0016H	1 Byte	Media descriptor.
0017H	1 Word	Number of FAT sectors.
0019H	1 Word	Sectors per track.
001BH	1 Word	Number of heads.
001DH	1 Word	Number of hidden sectors.
001FH	478 Bytes	Boot code.
01FEH	1 Word	55AAH signature word.

The first physical sector (cylinder 0, head 0, sector 1) of the hard disc contains the master boot record. The master boot record contains a code module and the disc partition table. The disc partition table contains the starting and ending cylinder of each of the disc partitions, as well as a flag that indicates whether the partition is active or not. Table 9-4 contains a description of the master boot record.

Table 9-4. Hard Disc Master Boot Record

Offset	Size	Description
0000H	446 Bytes	Master boot code.
01BEH	16 Bytes	Partition table entry #1.
01CEH	16 Bytes	Partition table entry #2.
01DEH	16 Bytes	Partition table entry #3.
01EEH	16 Bytes	Partition table entry #4.
01FEH	1 Word	0AA55H signature word.

A partition entry consists of 16 bytes. It contains information specifying the location of the partition, type of operating system, and a flag to indicate if the partition is active. Table 9-5 details the partition table entry.

Table 9-5. Partition Table Entry Record

Size	Description
1 Byte	Boot indicator.
1 Byte	Starting head number.
1 Byte	Starting sector number.
1 Byte	Starting cylinder number.*
1 Byte	System indicator.**
1 Byte	Ending head number.
1 Byte	Ending sector number.
1 Byte	Ending cylinder number.*
2 Words	Number of sectors in preceding partitions.
2 Words	Total number of sectors in partition.

* The actual cylinder number is a ten-bit value composed of the cylinder byte plus the two most significant bits of the associated sector byte. These two bits are the most significant bits of the ten-bit number.

** System indicators are:
00H = Unknown operating system
01H = DOS (12-bit FAT)
04H = DOS (16-bit FAT)

The INT 19H code will load the code module contained in the master boot record into memory, then transfer control to it. This code scans the data in the disc partition table to determine the active partition and its starting cylinder. The first sector of the active partition becomes the logical boot sector of the partition, and it contains a boot record. The boot record has the same format as the boot record contained on a flexible disc, except that some of the parameters are adjusted for the increased capacity of the hard disc partition. Refer to Table 9-3 for the format of a typical boot record.

A

BIOS Interrupts

This appendix includes three tables. The first lists the interrupt vector assignments. The second lists each of the STD-BIOS interrupts with supported functions. The third lists the EX-BIOS drivers; their vector addresses, functions and subfunctions.

Table A-1. Interrupt Vector Assignments

INT	Address	Function	Type/ Routine*	Service
0	000-003H	Divide by Zero	PI (1)	STD-BIOS
1	004-007H	Single Step	PI (1)	STD-BIOS
2	008-00BH	Nonmaskable Interrupt	PI	STD-BIOS
3	00C-00FH	Breakpoint	PI (1)	STD-BIOS
4	010-013H	Arithmetic Overflow	PI (1)	STD-BIOS
5	014-017H	Print Screen	SW (2)	STD-BIOS
6	018-01BH	Invalid Opcode	PI (1)	STD-BIOS
7	01C-01FH	Reserved	PI (1)	STD-BIOS
8	020-023H	Timer Interrupt	HW	
9	024-027H	Keyboard ISR (IRQ 1)	HW	STD-BIOS
A	028-02BH	Reserved (IRQ 2)	HW	STD-BIOS
B	02C-02FH	Serial Port 1 ISR (IRQ 3)	HW (1)	STD-BIOS
C	030-033H	Serial Port 0 ISR (IRQ 4)	HW (1)	STD-BIOS
D	034-037H	Printer Port 1 ISR (IRQ 5)	HW (1)	STD-BIOS
E	038-03BH	Flexible Disc ISR (IRQ 6)	HW	STD-BIOS
F	03C-03FH	Printer Port 0 ISR (IRQ 7)	HW (1)	STD-BIOS
10	040-043H	Video	SW (2)	STD-BIOS
11	044-047H	Equipment Check	SW (2)	STD-BIOS
12	048-04BH	Memory Size	SW (2)	STD-BIOS
13	04C-04FH	Flexible Disc/ Hard Disc	SW (2)	STD-BIOS
14	050-053H	Serial	SW (2)	STD-BIOS
15	054-057H	System Functions	SW (2)	STD-BIOS
16	058-05BH	Keyboard	SW (2)	STD-BIOS

Table A-1. Interrupt Vector Assignments (Cont.)

INT	Address	Function	Type/ Routine*	Service
17	05C-05FH	Printer	SW (2)	STD-BIOS
18	060-063H	Reserved	SW (3)	STD-BIOS
19	064-067H	Boot	SW (2)	STD-BIOS
1A	068-06BH	Time and Date	SW (2)	STD-BIOS
1B	06C-06FH	Keyboard Break	SW (3)	STD-BIOS
1C	070-073H	Timer Tick	SW (3)	STD-BIOS
1D	074-077H	Video Parameter Table	PT	STD-BIOS
1E	078-07BH	Flexible Disc Parameter Table	PT	STD-BIOS
1F	07C-07FH	Graphics Character Table	PT	STD-BIOS
20	080-083H	Program Terminate	SW	DOS
21	084-087H	DOS Function Calls	SW	DOS
22	088-08BH	DOS Terminate Address	PT	DOS
23	08C-08FH	DOS <Ctrl>-<Break> Address	SW	DOS
24	090-093H	DOS Critical Error	SW	DOS
25	094-097H	DOS Absolute Disc Read	SW	DOS
26	098-09BH	DOS Absolute Disc Write	SW	DOS
27	09C-09FH	DOS Terminate Stay Resident	SW	DOS
28-32	0A0-0CBH	Reserved for DOS	SW	DOS
33	0CC-0CFH	Mouse (RAM driver)	SW (2)	N/A
34-3F	0D0-0FFH	Reserved for DOS	SW	DOS
40	100-103H	Alternate Flexible Disc	SW	STD-BIOS
41	104-107H	Hard Disc Parameter Table (0)	PT	STD-BIOS
42-45	108-117H	Reserved	SW	STD-BIOS
46	118-11BH	Hard Disc Parameter Table (1)	PT	STD-BIOS
47-5F	11C-17FH	Reserved	SW	STD-BIOS
60-67	180-19FH	Reserved for User Programs Programs	SW	N/A
68-6E	1A0-1BBH	Unused	SW	N/A
6F	1BC-1BFH	Default EX-BIOS Entry Point	SW (2)	EX-BIOS
70	1C0-1C3H	Real-time Clock ISR (IRQ 8)	HW	STD-BIOS

Table A-1. Interrupt Vector Assignments (Cont.)

INT	Address	Function	Type/ Routine*	Service
71	1C4-1C7H	SW Redirected (IRQ 9)	HW	STD-BIOS
72	1C8-1CBH	Reserved (IRQ 10)	HW (1)	STD-BIOS
73	1CC-1CFH	Reserved (IRQ 11)	HW (1)	STD-BIOS
74	1D0-1D3H	HP-HIL (default IRQ 12)	HW (1)	EX-BIOS
75	1D4-1D7H	Coprocessor (IRQ 13)	HW	STD-BIOS
76	1D8-1DBH	Hard Disc ISR (IRQ 14)	HW (1)	STD-BIOS
77	1DC-1DFH	Reserved (IRQ 15)	HW (1)	STD-BIOS
78-7F	1E0-1FFH	Not Used	SW	N/A
80-F0	200-3C3H	Reserved	SW	N/A
F1-FF	3C4-3FFH	Not Used	SW	N/A

* PI--Processor interrupt
HW--Hardware interrupt
SW--Software interrupt
PT--Interrupt vector used as pointer to data
N/A--Not applicable
(1) UI--Unused interrupt ISR
(2) DRVR--Application callable entry point
(3) IRET--Interrupt return

The Table A-2 lists the STD-BIOS interrupt vectors, their usage and, where appropriate, their functions.

Table A-2. STD-BIOS Interrupts and Functions

INT Hex	Function Value	Function Equate	Definition
00H			Divide by zero
01H			Single step
02H			Non-maskable interrupt
03H			Breakpoint
04H			Arithmetic overflow
05H			Print screen
06H			Invalid opcode
07H			Reserved
08H			Timer interrupt
09H			Keyboard ISR
0AH			Reserved
0BH			Serial port 1 ISR
0CH			Serial port 0 ISR
0DH			Printer port 1 ISR
0EH			Flexible Disc ISR
0FH			Printer port 0 ISR
10H		INT_VIDEO	Video
	00H	F10_SET_MODE	Set video mode
	01H	F10_SET_CURSIZE	Set cursor size
	02H	F10_SET_CURPOS	Set cursor position
	03H	F10_RD_CURPOS	Read cursor position
	04H	F10_RD_PENPOS	Read light pen position
	05H	F10_SET_PAGE	Set active display page
	06H	F10_SCROLL_UP	Scroll rectangle up
	07H	F10_SCROLL_DN	Scroll rectangle down
	08H	F10_RD_CHARATR	Read character and attribute at cursor position
	09H	F10_WR_CHARATR	Write character and attribute at cursor position
	0AH	F10_WR_CHARCUR	Write character at cursor position
	0BH	F10_SET_PALLET	Set color pallet
	0CH	F10_WR_PIXEL	Write pixel
	0DH	F10_RD_PIXEL	Read pixel
	0EH	F10_WR_CHARTEL	Write teletype character
	0FH	F10_GET_STMODE	Get video state and mode
	10H-12H		Reserved
			Write string functions
	1300H	F10_WRS_00	global attribute
	1301H	F10_WRS_01	global attribute, move cursor
	1302H	F10_WRS_02	individual attributes
	1303H	F10_WRS_03	individual attributes, move cursor
	6F00H	F10_INQUIRE	EX-BIOS present
	6F01H	F10_GET_INFO	Get video parameters
	6F02H	F10_SET_INFO	Set video parameters
	6F03H	F10_MOD_INFO	Modifies video parameters
	6F04H	F10_GET_RES	Report video resolution
	6F05H	F10_XSET_MODE	Set video resolution

Table A-2. STD-BIOS Interrupts and Functions (Cont.)

INT Hex	Function Value	Function Equate	Definition
11H		INT_EQUIPMENT	Equipment check
12H		INT_MEM_SIZE	Memory size
			-- Note: both hard and flexible discs share interrupt 13H --
13H		INT_DISC	Disc Functions
	00H	F13_RESET_DISC	Reset Disc
	01H	F13_RD_LSTATUS	Read status of last operation
	02H	F13_RD_SECTORS	Read sectors
	03H	F13_WR_SECTORS	Write sectors
	04H	F13_VR_SECTORS	Verify sectors
	05H	F13_FORMAT_FLEX	Format flexible disc track
	06H		Reserved
	07H	F13_FORMAT_HDISC	Format hard disc
	08H	F13_GET_HPARMS	Get hard disc parameters
	09H-0BH		Reserved
	0CH	F13_TRACK_SEEK	Seek to track
	0DH	F13_ALT_RESET	Alternate hard disc reset
	0EH-014		Reserved
	15H	F13_GET_DASD	Read disc type (DASD)
	16H	F13_CHG_STATUS	Get disc change line status
	17H	F13_SET_DASD	Set disc type for formatting
14H		INT_SERIAL	Serial
	00H	F14_INIT	Initialize serial port parameters
	01H	F14_XMIT	Send out one character
	02H	F14_RECV	Receive one character
	03H	F14_STATUS	Get serial port status
	6F00H	F14_INQUIRE	EX-BIOS present
	6F01H	F14_EXINIT	Initialize serial port (19.2 Kbaud)
	6F02H	F14_PUT_BUFFER	Write a buffer of data
	6F03H	F14_GET_BUFFER	Read a buffer of data
	6F04H	F14_TRM_BUFFER	Read a buffer of data, terminate on specified condition
15H		INT_SYSTEM	System functions
	00H		Unsupported (turn on cassette motor)
	01H		Unsupported (turn off cassette motor)
	02H		Unsupported (read data blocks)
	03H		Unsupported (write data blocks)
	80H	F15_DEVICE_OPEN	Device open
	81H	F15_DEVICE_CLOSE	Device close
	82H	F15_PROG_TERM	Program termination
	83H	F15_WAIT_EVENT	Event wait
	84H	F15_JOYSTICK	Joystick support
	85H	F15_SYS_REQ	System request key pressed
	86H	F15_WAIT	Wait fixed amount of time
	87H	F15_BLOCK_MOVE	Extended memory transfer

Table A-2. STD-BIOS Interrupts and Functions (Cont.)

INT Hex	Function Value	Function Equate	Definition
	88H	F15_GET_XMEM _SIZE	Get extended memory size
	89H	F15_ENTER_PROT	Switch to protected mode
	90H	F15_DEV_BUSY	Device busy hook
	8BH	F15_INT_COMPLETE	Set Interrupt Completed Flag
16H		INT_KBD	Keyboard
	00H	F16_GET_KEY	Read keycode from kybd buffer
	01H	F16_STATUS	Report status of keyboard buffer
	02H	F16_KEY_STATE	Get key modifier status
	03H	F16_SET_TYPE_RATE	Set typematic rates
	05H	F16_PUT_KEY	Put data into keyboard buffer
	10H	F16_GET_EXT	Read keycode from buffer (including extended keycodes)
	11H	F16_EXT_STATUS	Report extended keyboard status
	12H	F16_EXT_KEY_STATE	Get Extended Key Modifier status
	6F00H	F16_INQUIRE	EX-BIOS present
	6F01H	F16_DEF_ATTR	Report default typematic values
	6F02H	F16_GET_ATTR	Report typematic values
	6F03H	F16_SET_ATTR	Set typematic values
	6F04H	F16_DEF_MAPPING	Report default translator assignments
	6F05H	F16_GET_MAPPING	Report translator assignments
	6F06H	F16_SET_MAPPING	Set translator assignments
	6F07H	F16_SET_XLATORS	Set CCP and HP Function keys
	6F08H	F16_KBD	Report keyboard information
	6F09H	F16_KBD_RESET	Reset keyboard to defaults
	6F0AH	F16_READ_SPEED	Read current speed
	6F0BH	F16_SET_LOW_SPEED	Select machine's slowest speed
	6F0CH	F16_SET_HIGH_SPEED	Select machine's fastest speed
	6F0DH	F16_GET_INT_NUMBER	Return the current HPENTRY vector
17H		INT_PRINTER	Printer
	00H	F17_PUT_CHAR	Send printer one byte
	01H	F17_INIT	Initialize printer port
	02H	F17_STATUS	Get printer port status
	6F00H	F17_INQUIRE	EX-BIOS present
	6F01H		Reserved
	6F02H	F17_PUT_BUFFER	Write a buffer to printer port
	6F03H		Reserved
	6F04H		Reserved
	6F0FH	F16_SET_CACHE_ON	Turn cache on
	6F10H	F16_SET_CACHE_OFF	Turn cache off
	6F11H	F16_GET_CACHE_STATE	Get current cache state
	6F12H	F16_SET_MEDIUM_SPEED	Sets medium speed for cache machines
18H			Reserved
19H		INT_BOOT	Boot
1AH		INT_CLOCK	Time and date

Table A-2. STD-BIOS Interrupts and Functions (Cont.)

INT Hex	Function Value	Function Equate	Definition
	04H	F1A_GET_DATE	Read date from real-time clock
	05H	F1A_SET_DATE	Set date in real-time clock
	06H	F1A_SET_ALARM	Set alarm
	07H	F1A_RESET_ALARM	Reset alarm
1BH			Keyboard break
1CH			Timer tick
1DH			Video parameter table
1EH			Flexible disc parameter table
1FH			Graphics character table
20H			Program terminate
21H			DOS function calls
22H			DOS terminate address
23H			DOS <Ctrl>-<Break>n address
24H			DOS critical error
25H			DOS absolute disc read
26H			DOS absolute disc write
27H			DOS terminate stay resident
28H-32H			Reserved for DOS
33H		INT_HPMOUSE	Reserved for Mouse driver
34H-3FH			Reserved for DOS
40H			Alternate flexible disc
41H			Hard disc parameter table (0)
42H-45H			Reserved
46H			Hard disc parameter table (1)
47H-5FH			Reserved
60H-67H			Reserved for user programs
68H			Reserved
69H			Reserved
6AH			Reserved
6BH			Reserved
6CH			Reserved
6DH			Reserved
6EH			Reserved
6FH		HP_ENTRY (default)	Default EX-BIOS entry point
70H			Real-time Clock ISR (IRQ 8)
71H			SW redirected (IRQ 9)
72H			Reserved (IRQ 10)
73H			Reserved (IRQ 11)
74H			Reserved (IRQ 12)
75H			Coprocessor (IRQ 13)
76H			Hard disc ISR (IRQ 14)
77H			Reserved (IRQ 15)
78H-7FH			Not used
80H-F0H			Reserved
F1H-FFH			Not used

EX-BIOS Drivers and Functions

Many additional features of the HP system can be accessed through the software interrupt INT 6FH (EX-BIOS extensions, see Table A-3). To call the EX-BIOS extensions, the BP register must contain the vector address of the desired driver, the AH register must contain the function code, and the AL register must contain the subfunction code. The rest of the registers are available for passing data and returning data to and from the routine.

In general, the AX, BP and DS registers are not preserved. They must be preserved by the calling routine if it needs them. See Chapter 2 for an example showing how EX-BIOS drivers are called.

Table A-3. EX-BIOS Drivers and Functions

Vector Address	Func. Value	Function Equate	Definition
0000H		V_SCOPY	Copyright notice routine
0006H		V_DOLITTLE	NOP routine (IRET)
000CH		V_PNULL	Null device driver
0012H		V_SYSTEM	System management functions
0012H	00	F_ISR	Interrupt service routine (unsupported)
0012H	02	F_SYSTEM	Standard driver functions
0012H	02/00	SF_INIT	System initialization
0012H	04	F_INS _BASEHPVT	Return HP_VECTOR_TABLE segment
0012H	06	F_INS _XCHGFIX	Exchange fixed table entries
0012H	08	F_INS _XCHGRSVD	Set next "reserved" entry in table
0012H	0A	F_INS _XCHGFREE	Set next "free" entry in table
0012H	0C	F_INS _FIXOWNDS	Install fixed vector, user supplied DS
0012H	0E	F_INS _FIXGETDS	Install fixed vector, system supplies DS
0012H	10	F_INS _FIXGLBDS	Install fixed vector, DS set to global data area
0012H	12	F_INS _FREEOWNDS	Install next free vector, user supplies DS
0012H	14	F_INS _FREEGETDS	Install next free vector, system supplies DS
0012H	16	F_INS _FREEGLBDS	Install next free vector, DS set to global data area
0012H	18	F_INS_FIND	Search for matching device header
0012H	1A		Reserved*
0012H	1C		Reserved*
0012H	1E	F_RAM_GET	Get EX-BIOS memory pool address and size
0012H	20	F_RAM_RET	Set memory pool address and size

Table A-3. EX-BIOS Drivers and Functions (Cont.)

Vector Address	Func. Value	Function Equate	Definition
0012H	22	F_CMOS_GET	Read and verify CMOS memory
0012H	24	F_CMOS_RET	Write to CMOS memory
0012H	26		Reserved*
0012H	28		Reserved*
0012H	2A	F_YIELD	Just returns
0012H	2C		Reserved*
0012H	2E		Reserved*
0012H	30	F_SND_CLICK _ENABLE	Enable keyclick
0012H	32	F_SND_CLICK _DISABLE	Disable keyclick (Default)
0012H	34	F_SND_CLICK	Execute keyclick if enabled
0012H	36	F_SND_BEEP _ENABLE	Enable beep
0012H	38	F_SND_BEEP _DISABLE	Disable beep
0012H	3A	F_SND_BEEP	Beep if enabled
0012H	3C	F_SND_SET _BEEP	Set beep frequency
0012H	3E	F_SND_TONE	Produce tone, user supplied duration and frequency
0012H	40	F_STR_GET _FREE_INDEX	Return next free string index
0012H	42	F_STR_DEL _BUCKET	Delete bucket string list
0012H	44	F_STR_PUT _BUCKET	Add bucket to current string list
0012H	46	F_STR_GET _STRING	Search the list for index, return string
0012H	48	F_STR_GET _INDEX	Search list for a string, return index
0018H			Reserved*
001EH		V_S8259	8259 interrupt controller support
001EH	00	F_ISR	Unsupported
001EH	02	F_SYSTEM	System functions
001EH	02/00	SF_INIT	Initialize HP-HIL IRQ
001EH	02/02	SF_START	Enable HP-HIL interrupts
001EH	02/06	SF_VERSION _DESC	Report HP version number
001EH	04	F_IO_CONTROL	Entry point to I/O control functions
001EH	04/00	SF_ENABLE _SVC	Unmask svc/8041 interrupt
001EH	04/02	SF_DISABLE _SVC	Mask svc/8041 interrupt
001EH	04/04	SF_ENABLE _KBD	Unmask keyboard INT 9 interrupt
001EH	04/06	SF_DISABLE _KBD	Mask keyboard INT 9 interrupt
001EH	04/08	SF_ENABLE _HPHIL	Unmask HP-HIL interrupt
001EH	04/0A	SF_DISABLE _HPHIL	Mask HP-HIL interrupt
0024H			Reserved*
002AH		V_SINPUT	Inquire Commands
002AH	00	F_ISR	Pass ISR event record to physical driver
002AH	02	F_SYSTEM	System functions
002AH	02/00	SF_INIT	Supported

Table A-3. EX-BIOS Drivers and Functions (Cont.)

Vector Address	Func. Value	Function Equate	Definition
002AH	04	F_IO_CONTROL	Entry point to I/O control functions
002AH	04/00	SF_DEF _LINKS	Set header link fields to system defaults
002AH	04/02	SF_GET _LINKS	Return device header link field entries
002AH	04/04	SF_SET _LINKS	Set device header link field entries
002AH	06	F_INQUIRE	Return describe record for an HP-HIL device
002AH	08	F_INQUIRE_ALL	Return device IDs for all HP-HIL devices present
002AH	0A	F_INQUIRE _FIRST	Return vector address of first HP-HIL device driver
002AH	0C	F_REPORT _ENTRY	Report entry point of PGID
0030H			Reserved*
0036H		V_QWERTY	Typewriter keypad translator
0036H	00	F_ISR	Translate to PC scan code.
0036H	02	F_SYSTEM	System functions
0036H	02/06	SF_VERSION _DESC	Report HP version number
003CH		V_SOFTKEY	Physical HP function key translator
003CH	00	F_ISR	Translate to PC scan code
003CH	02	F_SYSTEM	System functions
003CH	02/00	SF_INIT	Driver initialization
003CH	02/06	SF_VERSION _DESC	HP version number
0042H		V_FUNCTION	Compatibility function key translator
0042H	00	F_ISR	Logical Interrupt
0042H	02	F_SYSTEM	System functions
0042H	02/06	SF_VERSION _DESC	Report HP version number
0048H		V_NUMPAD	Numeric keypad translator
0048H	00	F_ISR	Logical interrupt
0048H	02	F_SYSTEM	System functions
0048H	02/06	SF_VERSION _DESC	Report HP version number
004EH		V_CCP	HP cursor control keypad translator
004EH	00	F_ISR	Logical interrupt
004EH	02	F_SYSTEM	System functions
004EH	02/06	SF_VERSION _DESC	Report HP version number
0054H		V_SVIDEO	Video Functions
0054H	00	F_ISR	Interrupt service routine
0054H	02	F_SYSTEM	Standard driver functions
0054H	04	F_IO_CONTROL	Driver dependent control functions
0054H	04/00	SF_VID _ID_HP	Returns the value "HP" in BX register

Table A-3. EX-BIOS Drivers and Functions (Cont.)

Vector Address	Func. Value	Function Equate	Definition
0054H	04/02	SF_VID _GET_INFO	Return video display adapter information
0054H	04/04	SF_VID _SET_INFO	Set info. on extended control register of the Multimode Video Adapter
0054H	04/06	SF_VID _MOD_INFO	Modify extended control register of Multimode Video Adapter
0054H	04/08	SF_VID _GET_RES	Get the resolution of active video adaptor
0054H	04/0A	SF_VID _SET_MODE	Set video mode of active Display adapter
005AH		V_STRACK	Sprite control
005AH	00	F_ISR	Update sprite
005AH	02	F_SYSTEM	System functions
005AH	02/00	SF_INIT	Initialize driver
005AH	02/02	SF_START	Start driver
005AH	04	F_TRACK_INIT	Set tracking to default state
005AH	06	F_TRACK_ON	Enable tracking
005AH	08	F_TRACK_OFF	Disable tracking
005AH	0A	F_DEF_MASKS	Define sprite masks
005AH	0C	F_SET_LIMITS_X	Set max/min horizontal values
005AH	0E	F_SET_LIMITS_Y	Set max/min vertical values
005AH	10	F_PUT_SPRITE	Display sprite
005AH	12	F_REMOVE_SPRITE	Remove sprite from display
0060H		V_EVENT_TOUCH	Application access to touch events
0066H		V_EVENT_TABLET	Application access to tablet events
006CH		V_EVENT_POINTER	Application access to pointer events
0072H -84H			Reserved*
008AH		V_CCPCUR	HP cursor control keypad translator
008AH	00	F_ISR	Logical interrupt
008AH	02	F_SYSTEM	System functions
008AH	02/06	SF_VERSION_DESC	Return HP version number
0090H		V_RAW	Return untranslated CCP data
0090H	00	F_ISR	Logical interrupt
0090H	02	F_SYSTEM	System functions
0090H	02/06	SF_VERSION_DESC	Return HP version number
0096H		V_CCPNUM	Translate scancodes from numeric keypad
0096H	00	F_ISR	Logical interrupt
0096H	02	F_SYSTEM	System functions
0096H	02/06	SF_VERSION_DESC	Return HP version number
009CH		V_OFF	Discard CCP and SOFTKEY scancodes

Table A-3. EX-BIOS Drivers and Functions (Cont.)

Vector Address	Func. Value	Function Equate	Definition
009CH	00	F_ISR	Logical Interrupt.
009CH	02	F_SYSTEM	System functions
009CH	02/06	SF_VERSION_DESC	Returns HP version number
00A2H		V_CCPGID	Translate CCP data to T_REL16 data
00A8H		V_SKEY2FKEY	HP and compatibility function key translator
00A8H	00	F_ISR	Logical interrupt
00A8H	02	F_SYSTEM	System functions
00A8H	02/06	SF_VERSION_DESC	Return HP version number
00AEH		V_8041	8041/keyboard interface. provides HP extensions to INT 16H
00AEH	00	F_ISR	Process ISR event record
00AEH	02	F_SYSTEM	System functions
00AEH	02/00	SF_INIT	Initialize driver
00AEH	02/02	SF_START	Driver start-up
00AEH	02/06	SF_VERSION_DESC	Report HP version number
00AEH	04	F_IO_CONTROL	Driver Dependant Functions
00AEH	04/00 through 04/08		Reserved*
00AEH	04/0A	SF_CREAT_INTR	Create interval entry
00AEH	04/0C	SF_DELET_INTR	Delete interval entry
00AEH	04/0E	SF_ENABL_INTR	Enable interval
00AEH	04/10	SF_DISBL_INTR	Disable interval
00AEH	04/12	SF_SET_RAMSW	Set RAM switch to one (1)
00AEH	04/14	SF_CLR_RAMSW	Set RAM switch to zero (0)
00AEH	04/16	SF_SET_CRTSW	Set CRT switch to one (1)
00AEH	04/18	SF_CLR_CRTSW	Set CRT switch to zero (0)
00AEH	04/1A	SF_PASS_THRU	Pass data byte to 8042
00AEH	04/1C through 04/2E		Reserved*
00B4H		V_PGID_CCP	Translate GID info to cursor control keypad format
00BAH		V_LTABLET	Application interface to tablet
00BAH	00	F_ISR	Logical interrupt
00BAH	02	F_SYSTEM	System functions
00BAH	02/00	SF_INIT	Initialize the driver data area
00BAH	02/02	SF_START	Start driver
00BAH	02/04	SF_REPORT_STATE	Report state of device
00BAH	02/06	SF_VERSION_DESC	Report driver version number
00BAH	02/08	SF_DEF_ATTR	Set default logical scaling attributes
00BAH	02/0A	SF_GET_ATTR	Get scaling attributes
00BAH	02/0C	SF_SET_ATTR	Set scaling attributes

Table A-3. EX-BIOS Drivers and Functions (Cont.)

Vector Address	Func. Value	Function Equate	Definition
00BAH	04	F_IO_CONTROL	I/O control functions
00BAH	04/00	SF_LOCK	Unsupported
00BAH	04/02	SF_UNLOCK	Unsupported
00BAH	04/04	SF_TRACK _ON	Turn cursor track on
00BAH	04/06	SF_TRACK _OFF	Turn cursor track off
00BAH	04/08	SF_CREATE _EVENT	Establish a new routine to be called on logical device events
00BAH	04/0A	SF_EVENT _ON	Enable event call to parent driver
00BAH	04/0C	SF_EVENT _OFF	Disable event call to parent driver
00BAH	04/0E	SF_CLIPPING _ON	Enable logical device clipping
00BAH	04/10	SF_CLIPPING _OFF	Disable logical device clipping
00BAH	06	F_SAMPLE	Report absolute position of GID
00C0H		V_LPOINTER	Application interface to Pointer/Mouse
00C0H	00	F_ISR	Logical Interrupt
00C0H	02	F_SYSTEM	System functions
00C0H	02/00	SF_INIT	Initialize the driver data area
00C0H	02/02	SF_START	Start driver
00C0H	02/04	SF_REPORT _STATE	Report state of device
00C0H	02/06	SF_VERSION _DESC	Report driver version number
00C0H	02/08	SF_DEF _ATTR	Set default logical scaling attributes
00C0H	02/0A	SF_GET _ATTR	Get scaling attributes
00C0H	02/0C	SF_SET _ATTR	Set scaling attributes
00C0H	04	F_IO_CONTROL	I/O control functions
00C0H	04/00	SF_LOCK	Unsupported
00C0H	04/02	SF_UNLOCK	Unsupported
00C0H	04/04	SF_TRACK _ON	Turn cursor track on
00C0H	04/06	SF_TRACK _OFF	Turn cursor track off
00C0H	04/08	SF_CREATE _EVENT	Establish a new routine to be called on logical device events
00C0H	04/0A	SF_EVENT _ON	Enable event call to parent driver
00C0H	04/0C	SF_EVENT _OFF	Disable event call to parent driver
00C0H	04/0E	SF_CLIPPING _ON	Enable logical device clipping
00C0H	04/10	SF_CLIPPING _OFF	Disable logical device clipping
00C0H	06	F_SAMPLE	Report absolute position GID
00C6H		V_LTOUCH	Application interface to touchscreen
00C6H	00	F_ISR	Logical interrupt
00C6H	02	F_SYSTEM	System functions
00C6H	02/00	SF_INIT	Initialize the driver data area
00C6H	02/02	SF_START	Start driver
00C6H	02/04	SF_REPORT _STATE	Report state of device

Table A-3. EX-BIOS Drivers and Functions (Cont.)

Vector Address	Func. Value	Function Equate	Definition
00C6H	02/06	SF_VERSION _DESC	Report driver version number
00C6H	02/08	SF_DEF _ATTR	Set default logical scaling attributes
00C6H	02/0A	SF_GET _ATTR	Get scaling attributes
00C6H	02/0C	SF_SET _ATTR	Set scaling attributes
00C6H	04	F_IO_CONTROL	I/O control functions
00C6H	04/00	SF_LOCK	Unsupported
00C6H	04/02	SF_UNLOCK	Unsupported
00C6H	04/04	SF_TRACK _ON	Turn cursor track on
00C6H	04/06	SF_TRACK _OFF	Turn cursor track off
00C6H	04/08	SF_CREATE _EVENT	Establish a new routine to be called on logical device events
00C6H	04/0A	SF_EVENT _ON	Enable event call to parent driver
00C6H	04/0C	SF_EVENT _OFF	Disable event call to parent driver
00C6H	04/0E	SF_CLIPPING _ON	Enable logical device clipping
00C6H	04/10	SF_CLIPPING _OFF	Disable logical device clipping
00C6H	06	F_SAMPLE	Report absolute position of GID
0108H		V_NULL	No driver
010EH			Reserved *
0114H		V_HPHIL	Setup HP-HIL to INPUT driver linkage
0114H	00	F_ISR	Logical interrupt
0114H	02	F_SYSTEM	System functions
0114H	02/00	SF_INIT	Initialize the driver data area
0114H	02/04	SF_REPORT _STATE	Report state of device
0114H	02/06	SF_VERSION _DESC	Report driver version number
0114H	02/0E	SF_OPEN	Driver in open state
0114H	02/10	SF_CLOSE	Put driver in closed state
0114H	04	F_IO_CONTROL	I/O control to driver
0114H	04/06	SF_CRV _RECONFIGURE	Force HP-HIL to reconfigure all devices
0114H	04/08	SF_CRV _WR_PROMPTS	Write a prompt to a device
0114H	04/0A	SF_CRV _WR_ACK	Write an acknowledge to a device
0114H	04/0C	SF_CRV _REPEAT	Set either 30Hz or 60Hz repeat rate

Table A-3. EX-BIOS Drivers and Functions (Cont.)

Vector Address	Func. Value	Function Equate	Definition
0114H	04/0E	SF_CRV _DISABLE_REPEAT	Cancel keyboard repeat rate
0114H	04/10	SF_CRV _SELF_TEST	Issue self-test command to physical device
0114H	04/12	SF_CRV _REPORT_STATUS	Get status from any HP-HIL device that needs to report
0114H	04/14	SF_CRV _REPORT_NAME	Return the ASCII name for a device
0114H	04/20	SF_GET _DEVTBL	Gets physical device table address
0114H	04/22	SF_SET _DEVTBL	Sets physical device table address
0114H	04/24	SF_DEF _DEVTBL	Sets default physical device table
0114H	08	F_GET_BYTE	Read one byte from specified HP-HIL device
0114H	0A	F_PUT_BUFFER	Write a string of bytes to HP-HIL device
011AH-016DH			Reserved*
016EH		V_SCANDOOR	
016EH	00	F_ISR	Process SCANDOOR interrupt
016EH	02	F_SYSTEM	System function
016EH	02/00	SF_INIT	Initialize driver
016EH	02/02	SF_START	Driver start-up
016EH	02/06	SF_VERSION_DESC	Reports HP version number
016EH	04	F_IO_CONTROL	Driver-dependent function
016EH	08	F_STATE_IOCTL	State functions
016EH	08/00	SF_GET_STATE	Get a STATE byte
016FH-1C2H			
1C8H-228H			Vectors available (16)
xxxH**		HP-HIL driver vectors 1 thru 7	Physical HP-HIL driver vectors
	00	F_ISR	Logical interrupt
	02	F_SYSTEM	System functions
	02/00	SF_INIT	Initialize driver
	02/02	SF_START	Start driver
	02/04	SF_REPORT _STATE	Unsupported
	02/06	SF_VERSION _DESC	Report HP version number
xxxH**		Available Vectors	Inquiry on availability of free vector in HP_VECTOR _TABLE

* Vectors marked reserved should not be used.

** Vectors with addresses xxxH do not have a fixed location. Their location is determined at power-on, depending on the system's configuration.

B

Memory Map

System Memory Map

The system maintains ROM and RAM entry point compatibility with the industry standard. Table B-1 provides a memory map of the first megabyte of memory.

Table B-1. Memory Map

Description	Starting Address	Absolute Begin/End
Interrupt Vectors	0000:0000H	00000H 003FFH
STD-BIOS Data Area	0040:0000H	00400H 0051DH
Scratch	0050:001EH	0051EH 005FFH
Bios Stack	0060:0000H	00600H 006FFH
DOS	0070:0000H	00700H
Application	0C00:0050H	0C050H nF000H
EX-BIOS System RAM		nF000H nFFFFH n is dependent upon the amount of memory installed. The EX-BIOS takes a minimum of 1000 hex bytes.
If Max RAM Equal 256KB		00000H 3FFFFH
If Max RAM Equal 512KB		00000H 7FFFFH
If Max RAM Equal 640KB		00000H 9FFFFH
Boot Address	07C0:0000H	07C00H
Reserved Video Buffer	A000:0000H	A0000H AFFFFH
Monochrome Video Buffer	B000:0000H	B0000H B7FFFH
Color Video Buffer	B800:0000H	B8000H BFFFFH
Video ROM Space	C000:0000H	C0000H C7FFFH
IHV ROM	C800:0000H	C8000H DFFFFH
SPU IHV ROM Space	E000:0000H	E0000H EFFFFH
BIOS ROM	F000:0000H	F0000H FFFFFH
RESET Vector	F000:FFF0H	FFFF0H

STD-BIOS Data Structures

The data area for the STD-BIOS is in absolute memory locations 00400H through 005FFH, which conforms to the industry standard. Table B-2 summarizes the assignments within this block of memory. A detailed description of these data fields follows the summary.

Table B-2. STD-BIOS Data Area

Address	Function
400H-407H	RS-232 Communication Port Addresses
408H-40FH	Parallel Printer Port Addresses
410H-416H	Equipment Flag
417H-43DH	Keyboard Data Area
43Eh-448H	Flexible Disc Data Area
449H-466H	Video Display Data Area
467H-46BH	Option ROM Data Area
46CH-470H	Timer Data Area
471H-473H	System Data Flags
474H-477H	Hard Disc Data Area
478H-47FH	Printer Timeout Counters
480H-483H	Keyboard Buffer Pointers
484H-488H	Enhanced Graphics Adapter (EGA) Data Area
489H-48AH	Reserved for Display Adapters
48BH-48BH	Flexible Disc Data Rate Area
48CH-48FH	Extended Hard Disc Data Area
490H-495H	Extended Flexible Disc Data Area
496H-497H	Keyboard Mode Indicator/LED Data Area
498H-4A0H	Real-Time Clock Data Area
4A1H-4A7H	Reserved for Network Adapter Cards
4A8H-4ABH	Pointer to EGA Data Area
4ACH-4EFH	Flexible Disc Expander adapter area (Vectra RS Only)
4F0H-4FFH	Intra-application Communications Area
500H-500H	Print Screen Status
501H-503H	Reserved
504H-504H	DOS Data Area
505H-5FFH	Reserved

RS-232 Communication Port Addresses

The I/O port addresses of up to four serial communication adapter ports are stored in these four words.

40:000H	02	S40_RS232_PORT1_ADR	port 1
40:002H	02	S40_RS232_PORT2_ADR	port 2
40:004H	02	S40_RS232_PORT3_ADR	port 3
40:006H	02	S40_RS232_PORT4_ADR	port 4

Parallel Printer Port Addresses

The I/O port addresses of up to four parallel printer adapter ports are stored in these four words.

40:008H	02	S40_PRINT_PORT1_ADR	port 1
40:00AH	02	S40_PRINT_PORT2_ADR	port 2
40:00CH	02	S40_PRINT_PORT3_ADR	port 3
40:00EH	02	S40_PRINT_PORT4_ADR	port 4

Equipment Byte Data Area

This data area contains several words describing some of the optional hardware installed in the system.

40:010H	02	S40_EQUIPMENT_FLAG	Installed devices word (see Table B-3)
40:012H	01	S40_MFG_INIT	Manufacturing initialization / test byte
40:013H	02	S40_MEMORY_SIZE	Memory size in 1k bytes
40:015H	01	S40_MFG_ERR_FLAG1	Manufacturing scratchpad
40:016H	01	S40_MFG_ERR_FLAG2	Manufacturing error codes

Table B-3. Equipment Flag (40:010H)

Bit	Value	Definition
0FH-0EH	0	no printers installed
	1	one printer installed
	2	two printers installed
	3	three printers installed
0DH-0CH		reserved
0BH-09H	0	no RS-232 ports installed
	1	one RS-232 port installed
	2	two RS-232 ports installed
	3	three RS-232 ports installed
	4	four RS-232 ports installed
08H		reserved

Table B-3. Equipment Flag (40:010H) (Cont.)

Bit	Value	Definition
07H-06H	0	1 flexible disc drive installed, if bit 0=1
	1	2 flexible disc drives installed, if bit 0=1
05H-04H	0	video adapter is not monochrome or color
	1	initial video mode of 40-column color
	2	initial video mode of 80-column color
	3	initial video mode of 80-column monochrome
03H-02H		reserved
01H	0	math coprocessor (80287 or 80387) not present
	1	math coprocessor (80287 or 80387) present
00H	0	no disc drives present
	1	some number of flexible disc drives present, see bits 7-6

Keyboard Data Area

This area is used by the keyboard driver to store keyboard states, scancodes and keycodes.

40:017H	01	S40_KBD_STATE1	State of special keys: shift, caps, etc. (see Table B-4).
40:018H	01	S40_KBD_STATE2	Secondary state of special keys (see Table B-5).
40:019H	01	S40_ALT_INPUT _ACCUM	Accumulator for alt/numpad entry
40:01AH	02	S40_KBD_BUF_HEAD	Keyboard buffer head pointer
40:01CH	02	S40_KBD_BUF_TAIL	Keyboard buffer tail pointer
40:01EH	20	S40_KBD_BUFFER	Keyboard buffer, room for 15 entries+overrun
40:096H	01	S40_KBD_EXT_STATE1	State of extended keyboard processing (see Table B-10).
40:097H	01	S40_KBD_STATUS	Keyboard LED status and data recieved from keyboard (see Table B-11).

Table B-4. Keyboard State Mask Byte1 (40:017H)

Bit	Data	Definition
07H	0	Insert state inactive
	1	Insert state active
06H	0	Caps lock state inactive
	1	Caps lock state active
05H	0	Num lock state inactive
	1	Num lock state active
04H	0	Scroll lock state inactive
	1	Scroll lock state active
03H	0	<Alt> key not depressed (inactive)
	1	<Alt> key depressed (active)
02H	0	<Ctrl> key not depressed (inactive)
	1	<Ctrl> key depressed (active)
01H	0	Left <Shift> key not depressed (inactive)
	1	Left <Shift> key depressed (active)
00H	0	Right <Shift> key not depressed (inactive)
	1	Right <Shift> key depressed (active)

Table B-5. Keyboard State Mask Byte2 (40:018H)

Bit	Data	Definition
07H	0	<Ins> key not depressed
	1	<Ins> key depressed
06H	0	<Caps lock> key not depressed
	1	<Caps lock> key depressed
05H	0	<Num lock> key not depressed
	1	<Num lock> key depressed
04H	0	<Scroll lock> key not depressed
	1	<Scroll lock> key depressed
03H	0	Pause state (<Ctrl>-<Num lock>) inactive
	1	Pause state active
02H	0	<System request> key not depressed
	1	<System request> key depressed
01H	0	left <Alt> key not depressed
	1	left <Alt> key depressed
00H	0	left <Ctrl> key not depressed
	1	left <Ctrl> key depressed

Flexible Disc Data Area

This area is used by the flexible disc driver to store information about current drive activity.

40:03EH	01	S40_FLOPPY _SEEK_STAT	Drive recalibration status (see Table B-6)
40:03FH	01	S40_FLOPPY _MOTOR_STAT	Drive motor status (see Table B-7)
40:040H	01	S40_FLOPPY _TIME_OUT	Drive timeout counter (see Table B-8)
40:041H	01	S40_FLOPPY _RETURN_STAT	Drive return code/error status
40:042H	07	S40_FLOPPY _CONTRL_STAT	Controller status/hard disc command/parm port copies

Table B-6. Flexible Disc Seek Status Byte (40:03EH)

Bit	Data	Definition
07H	1	Disc hardware interrupt occurred
06H-02H		Reserved
01H	0	Indicates drive 1 needs recalibration before next seek
	1	Indicates drive 1 does not need recalibration before next seek
00H	0	Indicates drive 0 needs recalibration before next seek
	1	Indicates drive 0 does not need recalibration before next seek

Table B-7. Flexible Disc Motor Status Byte (40:03FH)

Bit	Data	Definition
07H	0	Current operation is not a write
	1	Current operation is a write
06H		Reserved
05H	0	Drive one is not selected
	1	Drive one is selected
04H	0	Drive zero is not selected
	1	Drive zero is selected
03H-02H		Reserved
01H	0	Drive one motor is not running
	1	Drive one motor is running
00H	0	Drive zero motor is not running
	1	Drive zero motor is running

Table B-8. Flexible Disc Drive Error Status (40:041H)

Bit	Data	Definition
07H	1	Timeout error; disc failed to respond in time
06H	1	Seek error; seek to track failed
05H	1	Controller error; disc controller chip failed
04H-00H	1	Bad command; invalid command request
	2	Address error; address mark on disc not found
	3	Write protect error
	4	Sector not found; unable to locate sector, disc damaged or unformatted
	6	Media changed; the drive door was opened on a 1.2MB disc drive
	8	DMA error; DMA failed to respond in time
	9	Segment wrap; attempt to perform DMA across a segment boundary
	10H	CRC error; CRC check on data failed

Video Display Data Area

This area is used by the video driver to store current screen parameters and cursor positions.

40:049H	01	S40_CRT_MODE	Current video mode
40:04AH	02	S40_CRT_WIDTH	Current # of screen columns
40:04CH	02	S40_CRT_LENGTH	Current length of screen in bytes
40:04EH	02	S40_CRT_PAGE_ADR	Current address of current display page
40:050H	10	S40_CRT_CURSOR _POS	Cursor coordinates (row, column) up to 8 pages
40:060H	02	S40_CRT_CURSOR _MODE	Current cursor mode setting
40:062H	01	S40_CRT_DISPLAY _PAGE	Current display page
40:063H	02	S40_CRT_PORT_ADR	Base I/O port address for active video controller
40:065H	01	S40_CRT_MODE _SEL_REG	Mode select register copy
40:066H	01	S40_CRT_PALETTE	Color palette register copy

Option ROM Data Area

This area is used by the POST (SYSGEN) routine.

40:067H	02	S40_XROM_INIT_ADR	Offset address for optional I/O ROM initialization routine
40:069H	02	S40_XROM_SEGMENT	Segment address for optional I/O ROM
40:06BH	01	S40_XROM_INT_FLAG	Flag last interrupt that occurred

Timer Data Area

This area stores the current timer count and flags.

40:06CH	02	S40_TIMR_LOW	Least significant word of timer count
40:06EH	02	S40_TIMR_HIGH	Most significant word of timer count
40:070H	01	S40_TIMR_OVR_FLOW	24-hour timer tick rollover counter

System Data Flags

This area used by the system to flag <Ctrl>-<Break> and <Ctrl>-<Alt>-<DEL> requests.

40:071H	01	S40_SYS_BREAK_FLAG	System break request flag
40:072H	02	S40_SYS_RESET_FLAG	System reset flag

Hard Disc Data Area

This area is used by the INT 13H fixed disc driver to store current information about the fixed disc controller and status.

40:074H	01	S40_FD_STATUS	Hard disc status of last Int 13H operation
40:075H	01	S40_FD_COUNT	Number of hard discs present
40:076H	01	S40_FD_CONTROL	Copy of hard disc controller register
40:077H	01	S40_FD_PORT_OFFSET	Hard disc port offset

Printer Timeout Counters

These tables contain timeout counts for the parallel and serial ports. The default value is 14H for the parallel printer port and 01H for the serial port.

40:078H	01	S40_PRINT_TIMEOUT1	Parallel port 1 timeout count
40:079H	01	S40_PRINT_TIMEOUT2	Parallel port 2 timeout count
40:07AH	01	S40_PRINT_TIMEOUT3	Parallel port 3 timeout count
40:07BH	01	S40_PRINT_TIMEOUT4	Parallel port 4 timeout count
40:07CH	01	S40_RS232_TIMEOUT1	Serial port 1 timeout count
40:07DH	01	S40_RS232_TIMEOUT2	Serial port 2 timeout count
40:07EH	01	S40_RS232_TIMEOUT3	Serial port 3 timeout count
40:07FH	01	S40_RS232_TIMEOUT4	Serial port 4 timeout count

Keyboard Buffer Pointers

These pointers indicate where in memory the keyboard buffer is, as opposed to the current access points to the buffer stored in the pointers above. This allows an application to move and enlarge the keyboard buffer.

40:080H	02	S40_KBD_BUF_START	Pointer to physical start of keyboard buffer
40:082H	02	S40_KBD_BUF_END	Pointer to physical end of keyboard buffer

Enhanced Graphics Adapter (EGA) Data Area

This data area is used by the optional EGA driver when present.

40:084H	01	S40_EGA_CRT_ROW _CNT	Number of CRT rows minus one
40:085H	02	S40_EGA_CHAR_SIZE	Number of bytes per character in font table
40:087H	01	S40_EGA_INFO1 EGA	miscellaneous information
40:088H	01	S40_EGA_INFO2 EGA	miscellaneous information
40:089H	02		Reserved

Flexible Disc Data Rate Area

This data area is used by the flexible disc driver to optimize performance on the 1.2 MB drives by keeping track of the last data rate selected for disc access.

40:08BH	01	S40_FLOPPY_LAST _RATE	Last data rate selected

Extended Hard Disc Data Area

40:08CH	01	S40_AFD_STATUS_REG	Hard disc status reg. copy
40:08DH	01	S40_AFD_ERROR_REG	Hard disc error reg. copy
40:08EH	01	S40_AFD_INTR_FLAG	Hard disc interrupt flag

Extended Flexible Disc Data Area

This data area is used by the flexible disc driver to store information about the current media in the drives and what operations are being performed on it.

40:08FH	01	S40_AFLOPPY_INDICATORS	Drive 0 and 1 indicator flags
40:090H	01	S40_AFLOPPY_MEDIA	Drive 0 media state (see Table B-9)
40:091H	01	S40_AFLOPPY_MEDIA1	Drive 1 media state
40:092H	01	S40_AFLOPPY_OPER0	Drive 0 operation state
40:093H	01	S40_AFLOPPY_OPER1	Drive 1 operation state
40:094H	01	S40_AFLOPPY_TRACK0	Drive 0 current track
40:095H	01	S40_AFLOPPY_TRACK1	Drive 1 current track

Table B-9. Flexible Disc Media Byte (40:090H)

Bit	Data	Definition
07H-06H	0	Data transfer rate is 500 KB/sec
	1	Data transfer rate is 300 KB/sec
	2	Data transfer rate is 250 KB/sec
05H	0	Single step all seeks
	1	Double step all seeks
04H	0	Type of disc in drive unknown
	1	Type of disc in drive known
03H		Reserved
02H-00H	0	Attempting 360 KB disc in 360 KB drive
	1	Attempting 360 KB disc in 1.2 MB drive
	2	Attempting 1.2 MB disc in 1.2 MB drive
	3	Determined 360 KB disc in 360 KB drive
	4	Determined 360 KB disc in 1.2 MB drive
	5	Determined 1.2 MB disc in 1.2 MB drive

Keyboard Mode Indicator

This byte is used by the keyboard driver to store the current state of the keyboard LEDs.

40:096H	01	S40_KBD_EXT_STATE1	Keyboard LED flags (see Table B-10)
40:097H	01	S40_KBD_STATUS	Keyboard LED flags (see Table B-11)

Table B-10. 101-key Keyboard Flags (40:096H)

Bit	Data	Definition
07H	1	Read ID bytes in progress
06H	1	First of ID bytes was last
05H	1	Force Num Lock if 101-key keyboard is attached. This is when DOS is loaded or reloaded. Enhanced Keyboard only
04H	1	101-key keyboard attached. Enhanced Keyboard only
03H	1 1	Right <Alt> key status Right <Alt> key is pressed
02H	1 1	Right <Ctrl> key status Right <Ctrl> key is pressed
01H	1	E0 was last
00H	1	E1 was last

Table B-11. Keyboard LED Status and Data Area (40:097H)

Bit	Data	Definition
07H	1	Used for a flag to indicate 3 failures of sending data to keyboard
06H	1	LED update in progress
05H	1	Resend received from keyboard
04H	1	Acknowledge received from keyboard
03H	0	Reserved (set to 0)
02H	 1	Caps Lock LED status Caps Lock LED on
01H	 1	Num Lock LED status Num Lock LED on
00H	 1	Scroll Lock LED status Scroll Lock LED on

Note: Applications which modify these bytes may experience difficulty in maintaining synchronization between the Cursor Control keypad and the Numeric keypad on the HP Vectra Keyboard/DIN only.

Real-time Clock Data Area

This area is used by the RTC driver to store information needed to interrupt an application waiting on an RTC event.

40:098H	02	S40_RTC_WAIT _OFFSET	Offset address of user wait flag
40:09AH	02	S40_RTC_WAIT _SEGMENT	Segment address of user wait flag
40:09CH	02	S40_RTC_WAIT _CNT_LOW	Low word of wait count
40:09EH	02	S40_RTC_WAIT _CNT_HIGH	High word of wait count
40:0A0H	01	S40_RTC_WAIT _ACTV_FLG	Wait active flag
40:0A1H	07		Reserved

Pointer to EGA Data Area

40:0A8H	04	S40_EGA_TBL_PTR	Pointer to table of EGA pointers
40:0ACH	2C		Reserved

Flexible Disc Expander Adapter Data Area

This applies solely to the **Vectra RS** systems, and only when the Flexible Disc Expander adapter card is installed. This data area is used by the flexible disc expander driver to store information about the current media in the drives and what operations are being performed on it.

40:0D8H	01	S40_AFLOPPY_INDICATORS	Drive 2 and 3 indicator flags
40:0D9H	01	S40_AFLOPPY_MEDIA2	Drive 2 media state (see Table B-9)
40:0DAH	01	S40_AFLOPPY_MEDIA3	Drive 3 media state
40:0DBH	01	S40_AFLOPPY_OPER2	Drive 2 operation state
40:0DCH	01	S40_AFLOPPY_OPER3	Drive 3 operation state
40:0DDH	01	S40_AFLOPPY_TRACK2	Drive 2 current track
40:0DEH	01	S40_AFLOPPY_TRACK3	Drive 3 current track

Intra-application Communications Area

Used by applications to communicate with each other and with themselves from one work session to another.

40:0F0	10	S40_INTRA_APPL	Available to any application

Print Screen Status

40:100H	01	S40_PSCRN_STATUS	Flag for print screen in progress (see Table B-11)
40:101H	03		Reserved

Table B-12. Print Screen Status Byte (40:100H)

Bit	Data	Definition
07H-00H	0 1 FFH	Print not in progress Print in progress Error during print

DOS Data Area

The following data areas are used by DOS to provide status information on single-drive systems.

40:104H	01	S40_SINGLE_DRV _STAT	Status of flexible disc for single drive systems, i.e., currently drive A: or B:
40:105H	1A		Reserved

Reserved Data Areas

The following areas are reserved and should not be used under any circumstances:

40:089H	02
40:0A1H	07
40:0ACH	2C
40:0DFH	11
40:101H	03
40:105H	1A

EX-BIOS Data Area Map

Figure B-1 shows the EX-BIOS RAM space, which contains the HP_VECTOR_TABLE, the EX-BIOS memory pool, and the EX-BIOS global data area.

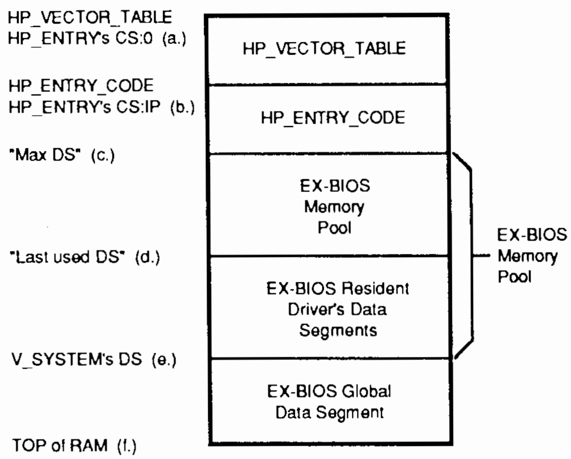


Figure B-1. EX-BIOS Data Area Layout

The following notes correspond to the letters in Figure B-1.

a. This address is the segment (CS) value stored in the second word of the HP_ENTRY interrupt vector (default 06FH); the HP_VECTOR_TABLE is at offset zero. This value may also be obtained from the V_SYSTEM driver, using function F_INS_BASEHPVT.

b. This address is the offset (IP) value stored in the first word of the HP_ENTRY interrupt vector (default 06FH). This address (CS:IP) represents the end of the HP_VECTOR_TABLE and points to the EX-BIOS's HP_ENTRY_CODE.

c. This address represents the last allocatable data segment ("MAX DS") value available from the EX-BIOS memory pool. This address may be obtained as well as allocated from the EX-BIOS V_SYSTEM driver. See F_RAM_GET and F_RAM_RET in Chapter 8.

d. This address is passed to drivers requesting memory from the EX-BIOS memory pool. Drivers must first subtract the size of their data segment from the "last used DS" value to get an addressable data area. The new "last used DS" is returned to the EX-BIOS using the F_RAM_RET function.

e. This address represents the EX-BIOS global data area used by drivers and services that share data. This address is the DS value stored in the HP_VECTOR_TABLE for the V_SYSTEM driver.

f. Top of RAM is the last address in memory. HP Vectra series of computers are shipped with 640KB of system memory, so this value is 9FFFFH. The data region between Top of RAM and the base of HP_VECTOR_TABLE is not directly available to applications. In the base system this region is 4KB long. However, since a user can reconfigure standard RAM in the Vectra series of computers (to 256KB, or 512KB via a jumper on the Processor PCA of the Vectra ES, and 512KB via a switch on the Processor PCA of the Vectra QS and RS), this region may need to be lengthened.

Option ROM Data Segments

An option ROM which does not have available on-board RAM can get memory in the manner described above. However, the problem arises as to how the option ROM is to 'remember' the data segment if it doesn't have any RAM to save the segment in. This problem usually can be solved since most option ROMs are accessed through the software interrupt mechanism. The option ROM adapter simply directs its entry point software interrupt vector to its EX-BIOS RAM data segment, which in turn jumps to the option ROM's entry point. That is,

```
CPU INT nn -> EX-BIOS data segment -> option ROM

     PUSH CS
     POP DS ; Load option ROM DS
     JMP FAR ROM_ENTRY_POINT
```

EX-BIOS Global Data Area

The EX-BIOS global data area is shared between several EX-BIOS drivers. It contains temporary and permanent variables required by the EX-BIOS to function properly. Some of these variables can be modified by application programs. As with the STD-BIOS data area, care should be taken when modifying the EX-BIOS data area.

The EX-BIOS global data area can be found by calling the V_SYSTEM driver, with the function F_INS_BASEHPVT. The EX-BIOS global data area segment will be returned in the DS register. Table B-13 defines the contents of this area.

Table B-13. EX-BIOS Global Data Area

Byte	Offset	Type	Definition
0-1DH	Reserved	Word	
1EH	T_STR_NEXT_INDEX	Word	The next unused string index number.
20H and up	Reserved		

ROM BIOS Memory Map

Table B-14 lists the compatible ROM entry points. The programmer should not access these entry points directly.

Table B-14. Rom Entry Points

Int	Rom Entry	Type	Function
2	F000:E2C3	code	Non-maskable interrupt
5	F000:FF54	code	Print screen
10	F000:F065	code	Video
11	F000:F84D	code	Equipment check
12	F000:F841	code	Memory size
13	F000:EC59	code	Flexible/hard disc
14	F000:E739	code	Serial
15	F000:F859	code	System functions
16	F000:E82E	code	Keyboard
17	F000:EFD2	code	Printer
18	F000:FF53	code	Reserved
19	F000:E6F2	code	Boot
1A	F000:FE6E	code	Time and date
1B	F000:FF53	code	Keyboard break
1C	F000:FF53	code	Timer tick
1D	F000:F0A4	data	Video parameter table
1E	0000:0522	data	Flexible disc parameter table
1F	F000:0000	data	Graphics character table

Product Identification

The following are Product Identification Strings. Application designers should use the product identification byte to differentiate among the various HP Vectra family of personal computers. The machine capability marker can be used to indicate a specific hardware or ROM BIOS capability which may apply across more than one product identification code.

ROM version independent information:

0F000:00F4H	DB	High Processor Clock Rate	Processor speed in MHz - exception: the value 0FFH means 8 MHz
0F000:00F5H	DB	Low Processor Clock Rate	
0F000:00F8H	DB	'HP'	HP Vectra PC ID
0F000:00FAH	DB	XXXXXXXXB	Product identification
0F000:00FBH	DB	XXXXXXXXB	Machine capability marker

ROM version dependent information:

0F000:00FCH	DW	PPSSH	Version number
0F000:00FEH	DB	YYH	ROM release year since 1960 stored in BCD
0F000:00FFH	DB	NN	Week of the year stored in BCD

Industry Standard PC ID:

0F000:FFFEH	DB	0FCH	IBM-AT Compatible PC

Product Identification Definitions

Processor Clock Rate

All Vectras (except for the original Vectra PC) set their clock rate bytes to their clock speeds in MHz. Machines which have a single clock rate should set both the primary and secondary rate bytes to the same value.

0F000:00F4H = High processor clock rate (primary)
Length = one byte

0F000:00F5H = Low processor clock rate (secondary)
Length = one byte

Processor Clock Rates for HP Vectra Series of Computers

Computer	Clock Rate (High)	Clock Rate (Low)
Vectra ES	08H (8 MHz)	08H (8 MHz)
Vectra ES/12	0CH (12 MHz)	08H (8 MHz)
Vectra QS/16, RS/16	10H (16 MHz)	08H (8 MHz)
Vectra QS/20, RS/20	14H (20 MHz)	08H (8 MHz)
Vectra RS/20C	14H (20 MHz)	05H (5 MHz)
Vectra RS/25C	19H (25 MHz)	05H (5 MHz)

HP Vectra PC ID

The HP Vectra PC ID flag is used to validate the ROM BIOS Identification Block. The flag should be tested prior to examining the other bytes of the block.

0F000:00FAH = Product Identification
Length = one byte

```
Bits:

7  6  5   4  3  2  1  0
-------   -------------
   |            |------>  00000 - Original Vectra PC
   |                      00001 - Vectra ES/12
   |                      00010 - Vectra RS/20
   |                      00011 - Portable Vectra CS
   |                      00100 - Vectra ES
   |                      00101 - Vectra CS
   |                      00110 - Vectra RS/16
   |                      00111 - Vectra QS/16
   |                      01000 - Vectra QS/20
   |                      01001 - Vectra RS/20C
   |                      01010 - Vectra RS/25C
   |                      01011 - Vectra LS/12
   |        01100 through 11111 - Reserved
   |
   |------------------->    000 - 80286
                            001 - 8088
                            010 - 8086
                            011 - 80386
                100 through 111 - Reserved
```

Machine Capability Marker

0F000:00FBH = Machine capability marker
Length = one byte

```
Bits:

7  6  5  4  3  2  1  0
-------------------   -
          |          |-->  1.44 MB flexible discs supported
          |                with automatic media sense
          |                reported via INT 13 function 8H.
          |
          |------------->  Reserved, set to 0.
```

BIOS Version Number

0F000:00FCH = BIOS version number
Length = two bytes

Encoding is as follows:

```
DW    PPSS
```

```
Where PP = Primary version number
      SS = Secondary version number
```

For example, BIOS release A.01.05 would be expressed as:

```
DW    0105
```

Note that when using DEBUG to look at the bytes, the numbers will be reversed (05 01).

Year of the ROM BIOS Release (in BCD)

0F000:00FEH = Year of ROM BIOS release in BCD.
Length = one byte

Encoded as follows:

```
DB    VV
```

Where VV is the difference of the current year and 1960, expressed in BCD.

For example, if the current year is 1987, the entry would be 1987 minus 1960 which is equal to 27H, expressed in BCD as:

```
DB    27H
```

Week of the ROM BIOS Release (in BCD)

0F000:00FFH = Week of the ROM BIOS release in BCD.
Length = one byte

Encoded as follows:

```
DB    NN
```

Where NN is the week in which the BIOS ROMs were released to manufacturing. The range is (00H - 51H), expressed in BCD.

For example, if the ROMs were released in week 15, the entry would be 15H in BCD, expressed as:

```
DB    15H
```

C

CMOS Memory Layout and Real-Time Clock

The real-time clock chip contains 64 bytes of non-volatile (CMOS) memory. Values saved in this memory area are not destroyed when the system is powered off. Table C-1 defines the use of the Real-time Clock and CMOS memory area.

Table C-1. CMOS Memory and Real-time Clock

CMOS Address	Application
00H	* RTC current second
01H	* RTC second alarm value
02H	* RTC current minute
03H	* RTC minute alarm value
04H	* RTC current hour
05H	* RTC hour alarm value
06H	* RTC current day of the week
07H	* RTC current day of the month
08H	* RTC current month
09H	* RTC current year
0AH	* RTC status register A
0BH	* RTC status register B
0CH	* RTC status register C
0DH	* RTC status register D
0EH	* Diagnostic status byte
0FH	* Shutdown status byte
10H	Flexible disc drive type (A: and B:)
11H	Reserved
12H	Hard Disc Type for drives C: and D: (1 through 14)
13H	Reserved
14H	Equipment byte
15H	Low base memory
16H	High base memory
17H	Extended memory size (low byte)
18H	Extended memory size (high byte)
19H	Extended Hard Disc Type for drive C: (16 through 255)
1AH	Extended Hard Disc Type for drive D: (16 through 255)
1BH-2DH	Reserved
2E-2FH	2-byte industry standard CMOS checksum for bytes 10H to 2DH

Table C-1. CMOS Memory and Real-time Clock (Cont.)

CMOS Address	Application
30H	* Extended memory size (low byte, defined by POST)
31H	* Extended memory size (high byte)
32H	* Date century byte
33H	* Information flags
34-3FH	* Reserved
40H-7FH	* Reserved

* These bytes are not included in the industry standard CMOS checksum

Real-Time Clock/CMOS Access

Port 70H and port 71H provide the interface to the real-time clock and CMOS memory controller. Port 70H is used to specify the byte address to read or write. Port 71H is used to pass the data. For example, to read the equipment byte, the programmer would write 14H to port 70H, then read the data byte from port 71H. A read or write to port 71H must always be preceeded by a write to port 70H.

Real-Time Clock (CMOS Address 00H-0DH)

The real-time clock (RTC) chip maintains the current date and time, even when the system is powered off. Four registers are initialized by the SETUP program when the user sets the current date and time. These registers are detailed in Tables C-2, C-3, C-4 and C-5.

Table C-2. CMOS_RTC_REGA (CMOS Address 0AH)

Bit	Data	Definition
7	0	The current date and time is available to read
	1	The current date and time are not available to read because an update of these values is in progress
6-4		Time divider selection bits to indicate what time base frequency is being used. This field is set to 2H to indicate that a 32.768 kHz crystal is providing the time base.
3-0		Rate selection bits to specify output square wave frequency. This field is set to 06H to select a square wave frequency of 1.024 kHz, or a periodic interrupt rate of 976.562 microseconds.

Table C-3. CMOS_RTC_REGB (CMOS Address 0BH)

Bit	Data	Definition
7	0	Update clock normally (default)
	1	Suspend clock updates
6	0	Disable periodic interrupts (default)
	1	Enable periodic interrupts
5	0	Disable alarm interrupts (default)
	1	Enable alarm interrupts
4	0	Do not generate an interrupt when the current update cycle completes (default)
	1	Generate an interrupt each time a clock update completes
3	0	Disable square wave output (default)
	1	Enable square wave output
2	0	Store date and time in BCD (Binary Coded Decimal) (default)
	1	Store date and time as binary integers
1	0	Places hours byte in 12-hour mode
	1	Places hours byte in 24-hour mode (default)
0	0	Disable daylight savings (default)
	1	Enable daylight savings

Table C-4. CMOS_RTC_REGC (CMOS Address 0CH)

Bit	Data	Definition
7	0	No interrupts are currently asserted
	1	The RTC is asserting an interrupt due to either the alarm, periodic interrupt, or update ended.
6	0	No periodic interrupt has occurred since the last read of this bit.
	1	A periodic interrupt has occurred, read only and cleared by read.
5	0	No alarm interrupt has occurred since the last read of this bit.
	1	An alarm interrupt has occurred, read only and cleared by read.
4	0	No update ended interrupt has occurred since the last read of the bit.
	1	An update ended interrupt has occurred, read only and cleared by read.
3-0		Reserved

Table C-5. CMOS_RTC_REGD (CMOS Address 0DH)

Bit	Data	Definition
7	0 1	Power was lost to the RTC chip since the last read of this bit. The RTC chip has not lost power since the last read of this bit. Read only, set to 1 after read.
6-0		Reserved

Diagnostic Status Byte (CMOS Address 0EH)

This byte is set by the POST routine to flag errors detected during power on. The contents of this byte are detailed in Table C-6.

Table C-6. CMOS_DIAGNOSTIC_STATUS (CMOS Address 0EH)

Bit	Data	Definition
7	1	Power to RTC failed
6	1	Bad industry standard CMOS checksum
5	1	Configuration inconsistency
4	1	Memory size does not match
3	1	Hard disc failed initialization
2	1	Invalid CMOS
1-0		Reserved

System Shutdown Byte (CMOS Address 0FH)

This byte is used by the system power-on sequence to determine what action is to be taken upon return from protected mode. The details of this byte are shown in Table C-7.

Table C-7. CMOS_SHUTDOWN_BYTE (CMOS Address 0FH)

Bit	Data	Definition
7-0	0-3	Perform power-on reset sequence
	4	INT 19H (reboot)
	5	Flush keyboard and jump indirect via double word 40:67H
	6-7	Reserved
	8	Used by POST during test of protected mode RAM
	9	Used for INT 15H support (block move)
	A	Jump indirect via double word at 40:67H
	B-FF	(Same as values 0-3)

Flexible Disc Descriptor Byte (CMOS Address 10H)

This byte is initialized by the SETUP program and indicates what types of flexible disc drives are installed. The details of this byte are shown in Table C-8.

Table C-8. CMOS_FDC_TYPE (CMOS Address 10H)

Bit	Data	Definition
7-4	0	No drive installed as drive A
	1	360 KB drive installed as drive A
	2	1.2 MB drive installed as drive A
	4	3.5-inch drive installed as drive A
3-0	0	No drive installed as drive B
	1	360 KB drive installed as drive B
	2	1.2 MB drive installed as drive B
	4	3.5-inch drive installed as drive B

CMOS Hard Disc Type (CMOS Address 12H)

CMOS_FIXED_ DISC_TYPE, (CMOS Address 12H), defines the type of the first and second hard disc drive installed.

00000000 through 00001111 define Hard Disc Drive Types 1 through 14. See Chapter 7 for more information.

Equipment Byte (CMOS Address 14H)

This byte is used to initialize STD-BIOS RAM location 40:0010H. This is the value returned by the STD-BIOS interrupt INT 11 (get current equipment). The details of this byte are shown in Table C-9.

Table C-9. CMOS_EQ_BYTE (CMOS Address 14H)

Bit	Data	Definition
7-6	0	One drive installed
	1	Two drives installed
5-4	1	Primary display is 40 column color
	2	Primary display is 80 column color
	3	Primary display is 80 column monochrome
3-2		Reserved
1	1	80287 or 80387 installed
0	1	At least one flexible disc installed

System Base Memory Size (CMOS Address 15H-16H)

This value represents the amount of base (DOS-addressable) memory installed in the system minus the amount of RAM used by the EX-BIOS data area. Three base memory configurations are valid:

0100H = 256 KB of base memory installed (Vectra ES series only)
0200H = 512 KB of base memory installed
0280H = 640 KB of base memory installed

Note that Vectra series of personal computers are shipped with 640 KB of base memory; however, these systems may be configured via jumpers or switches to the lower amounts listed.

The actual stored value will be adjusted to leave space for the EX-BIOS data area. For example, the value may be 00FCH instead of 0100H, indicating that the system is configured for 256 KB of base memory but the EX-BIOS data area is using 4 KB of it.

CMOS_BASE_MEMORY_LSB (CMOS Address = 15H)
CMOS_BASE_MEMORY_MSB (CMOS Address = 16H)

System Extended Memory Size (CMOS Address 17H-18H)

These values are initialized by the SETUP program to the user specified Extended memory size from zero to 15 MB in 512 KB increments. For example:

0200 = 512 KB of Extended memory (0.5 MB)
0400 = 1024 KB of Extended memory (1.0 MB)
0600 = 1536 KB of Extended memory (1.5 MB)
through
3A00 = 14848 KB of Extended memory (14.5 MB)
3C00 = 15360 KB of Extended memory (15.0 MB)

Note that Extended memory is memory above one megabyte.

CMOS_EXT_MEMORY_LSB (CMOS Address = 17H)
CMOS_EXT_MEMORY_MSB (CMOS Address = 18H)

Extended Hard Disc Type for Drive C: (CMOS Address 19H)

Bit 7-0 defines the Hard Disc Type of the first hard disc installed (drive C:):

00010000 to 11111111 define types 16 through 255. (The *SETUP Program Guide* in your computer's *Setting Up Vectra* binder contains a table listing hard disc drive type characteristics.)

Extended Hard Disc Type for Drive D: (CMOS Address 1AH)

Bit 7-0 defines the Hard Disc Type of the second hard disc installed (drive D:):

00010000 to 11111111 define types 16 through 255. (The *SETUP Program Guide* in your computer's *Setting Up Vectra* binder contains a table listing hard disc drive type characteristics.)

STD-BIOS Checksum Word (CMOS Address 2EH-2FH)

This word contains the checksum which is used to verify the contents of the STD-BIOS CMOS data locations. This checksum is computed each time one of these locations is modified using an EX-BIOS CMOS function. If the EX-BIOS is not used for CMOS update then it is the programmer's responsibility to calculate and replace the STD-BIOS checksum.

CMOS_STD_BIOS_CRC = [10]+[11]+[12]+. . .+[2DH]: 16-bit carryout

Low and High Extended Memory Byte (CMOS Address 30H-31H)

These bytes reflect the total extended memory above the 1MB address space determined at POST. Extended memory size can be determined through system INT 15.

Address 30H, Low extended memory size: Bit 7-0.
Address 31H, High extended memory size: Bit 7-0.

Valid sizes are:

0200H = 512K of Extended memory. (0.5 MB)
0400H = 1024K of Extended memory. (1.0 MB)
0600H = 1536K of Extended memory. (1.5 MB)
through
3C00H = 15360 KB of Extended memory (15 MB, maximum)

Date Century Byte (CMOS Address 32H)

This byte reflects the value for the century expressed in the BCD.

BCD value for the century (BIOS interface to read and set): Bit 7-0.

Test Information Byte (CMOS Address 33H)

Bit seven of this byte is initialized by the boot process to indicate that 640 KB of base memory is installed. The details of this byte are shown in Table C-10.

Table C-10. CMOS_TEST_INFO (CMOS Address 33H)

Bit	Data	Definition
7	1	Indicates top 128K of base memory is installed
6-0		Reserved

D

I/O Port Map

Appendix D describes the I/O map of the system. Table D-1 lists the I/O map of all devices integrated in the **System Processing Unit** (SPU). Table D-2 lists the recommended I/O port assignments for devices in adapter cards. Subsequent sections in the appendix describe the SPU built-in devices individually. I/O devices in adapter cards are described fully in the *Vectra Accessories Technical Reference Manual*.

Table D-1. SPU I/O Map

I/O Address	Description
000-01FH	First DMA Controller (8237A)
020-03FH	Master Interrupt Controller (8259A)
040-05FH	Timer Controller (8254)
060H	Keyboard Buffer Full port
061H	SPU Control port
064H	Keyboard Output Buffer Full (OBF) port
068H	Keyboard Extended Command port
06C-06FH	HP-HIL Controller ports
070H	RTC address / NMI disable port
071H	RTC data
078H	Hard Reset NMI enable port
080-09FH	DMA Page Registers ports
0A0-0BFH	Slave Interrupt Controller (8259A)
0C0-0DFH	Second DMA Controller (8237A)
0F0H	Clear (80287 or 80387 only) Coprocessor port
0F1H	Reset (80287 or 80387 only) Coprocessor port
0F8-0FFH	Math (80287 or 80387 only) Coprocessor

Table D-2. Adapter I/O Map

I/O Address	Description
1F0-1F8H	Hard Disc controller
200-207H	Game I/O adapter
278-27FH	Parallel port 2
2E8-2EFH	Serial port 3 (COM4)
2F8-2FFH	Serial port 1 (COM2)
300-31FH	Prototype adapter card
320-323H	Reserved
378-37FH	Parallel port 1
380-38FH	SDLC, bisynch 2
3A0-3AFH	Bisynch 1
3B0-3B7H	Monochrome display adapter
3BC-3BFH	Monochrome display/Parallel adapter
3C0-3CFH	Enhanced Graphics adapter (EGA)
3D0-3DFH	Color Graphics adapter
3E8-3EFH	Serial port 2 (COM3)
3F0-3F7H	Flexible Disc controller
3F8-3FFH	Serial port 0 (COM1)

DMA Channel Controller

The SPU supports seven DMA channels using two Intel 8237A compatable DMA controllers in cascade mode. For each DMA channel there is a page register used to extend the addressing range of the channel to 16 MB. The only type of DMA transfer allowed is "I/O to memory". No "I/O to I/O" or "memory to memory" transfers are allowed due to the way the hardware is configured. For more detailed information on the 8237A DMA controllers see Intel's *The 8086 Family User's Manual*. Table D-3 summarizes how the DMA channels are allocated.

Table D-3. DMA Channel Allocation

Channel	First DMA controller (used for 8 bit transfers):
0	Spare.
1	Usually datacomm.
2	Flexible disc I/O.
3	Spare.
	Second DMA controller (used for 16 bit transfers):
4	Cascade from first DMA controller.
5	Spare.
6	Spare.
7	Spare.

I/O Port Addresses for DMA Controllers

Table D-4 shows the I/O port addresses for the page register and DMA controllers used when writing the DMA addresses.

Table D-4. I/O Port Addresses and Address Lines

		DMA page register I/O Ports:
Channel	**I/O Port**	**Address Lines**
0	087H	A23-A16
1	083H	A23-A16 byte transfers
2	081H	A23-A16
3	082H	A23-A16
4		Not connected
5	08BH	A23-A17
6	089H	A23-A17 word transfers
7	08AH	A23-A17
X	08FH	Used for RAM refresh
		DMA Controller I/O Ports:
Channel	**I/O port**	
0	000H 001H	address register (A15-A0) byte count register
1	002H 003H	address register (A15-A0) byte count register
2	004H 005H	address register (A15-A0) byte count register
3	006H 007H	address register (A15-A0) byte count register
4	0C0H 0C2H	address register (A16-A1) word count register
5	0C4H 0C6H	address register (A16-A1) word count register
6	0C8H 0CAH	address register (A16-A1) word count register
7	0CCH 0CEH	address register (A16-A1) word count register

Notes:

Channel 4 (first channel on the second DMA controller) is used to cascade the first DMA controller and it must not be programmed for DMA transfers.

Channels 5 through 7 are word-wide channels so the address lines used are A1 through A23. Address line A0 is always forced to zero. The address register on these channels provides address lines A16 through A1, and address lines A23 through A17 come from bits 7 through 1 of the page register. Bit 0 of the page register is not used. Care should be taken in making sure that the counts and addresses are in words rather than bytes.

Table D-5 lists I/O ports used for writing commands to the DMA controllers.

Table D-5. DMA Controller Command I/O Ports

Contrl. 1	Contrl. 2	I/O Write	I/O Read
0D0H	008H	Command Register	Status Register
0D2H	009H	Request Register	illegal
0D4H	00AH	Single Mask Register	illegal
0D6H	00BH	Mode Register	illegal
0D8H	00CH	Clear Byte Pointer Flag	illegal
0DAH	00DH	Master Clear Command	Temporary Register
0DCH	00EH	Clear Mask Command	illegal
0DEH	00FH	Multi-Mask Register	illegal

8259A Interrupt Controllers

The system has two 8259A interrupt controllers. They are arranged as a master interrupt controller and a slave that is cascaded through the master. Table D-6 shows the I/O ports for these interrupt controllers and how they are cascaded.

Table D-6. 8259A Interrupt Controller I/O Ports

Register Name	Master	Slave
Command Register	20H	0A0H
Interrupt Mask Register	21H	0A1H

Table D-7 shows how the master and slave controllers are connected. The Interrupt Requests (IRQ) are numbered sequentially starting with the master 8259 controller and followed by the slave.

Table D-7. 8259A Master to Slave Connections.

Master's IRQ	Interrupt Request Description
IRQ0(08H)	Timer
IRQ1(09H)	Keyboard OBF
IRQ2(0AH)<--[Slave IRQ]	Reserved
IRQ08(70H)	Real Time Clock
IRQ09(71H)	SW Redirected
IRQ10(72H)	Serial Port 2 (COM3)
IRQ11(73H)	Serial Port 3 (COM4)
IRQ12(74H)	Reserved
IRQ13(75H)	Coprocessor
IRQ14(76H)	Hard Disc
IRQ15(77H)	Reserved
IRQ3(0BH)	Serial Port 1 (COM2)
IRQ4(0CH)	Serial Port 0 (COM1)
IRQ5(0DH)	Parallel Port 2
IRQ6(0EH)	Flexible Disc
IRQ7(0FH)	Parallel Port 1

Note: The numbers in parentheses are the interrupt vector numbers generated by the IRQs.

The following example shows how the 8259A controllers are programmed on initialization:

```
     CLI  ; Disable interrupts
PROGRAM_MASTER:
     MOV AL, 11H ; ICW1: Initialization Command
     OUT 20H,AL
     JMP $+2
     MOV AL,08H ; Interrupt Vector Base at 08H
     OUT 21H,AL
     JMP $+2
     MOV AL,04H ; Define master with slave
     OUT 21H,AL ; at IRQ2
     JMP $+2
     MOV AL,01H ; 8086/88 Mode
     OUT 21H,AL
     JMP $+2

PROGRAM_STD_SLAVE:
     MOV AL, 11H ; ICW1: Initialization Command
     OUT 0A0H,AL
     JMP $+2
     MOV AL,70H ; Interrupt Vector Base at 70H
     OUT 0A1H,AL
     JMP $+2
```

```
MOV AL,02H ; Slave ID number
OUT 0A1H,AL
JMP $+2
MOV AL,01H ; 8086/88 Mode
OUT 0A1H,AL
JMP $+2
STI        ; Re-enable interrupts
```

8254 Timer Controller (I/O Ports 40H through 43H)

The system contains an Intel Programmable Interval Timer 8254. The timer controller consists of three separate timer channels; timer channels 0, 1 and 2. Channel 0 provides the BIOS with a programmable time interval. Channel 1 provides the memory refresh signal of the dynamic RAMs in the system. Channel 2 generates a fixed frequency envelope to the sound generation circuit.

WARNING

Timer channel 1 should not be used. Writing to this channel may cause loss of data in system memory.

The timer chip interfaces to the CPU via 4 I/O ports:

I/O Port	Function
040H	Counter data for timer 0.
041H	Counter data for timer 1.
042H	Counter data for timer 2.
043H	The control register for all three timers.

See Intel's *The 8086 Family User's Manual* for more details of the 8254 timer controller.

Keyboard Data Buffer (60H)

The keyboard data buffer is read by the CPU when the keyboard asserts the OBF interrupt. The OBF signal is automatically cleared when the data buffer is read. See Chapter 5 for more information about the keyboard data buffer.

SPU Control Port (61H)

The SPU Control Port (61H) is a bi-directional buffer which latches an assortment of unrelated signals. Table D-8 describes the bit values contained in this buffer.

Table D-8. PORT 61H Bit Values

		When the CPU reads port 61H:
Bit	**Data**	**SPU Status Port Definition**
7	1	Parity error in on-board system ram
6	1	I/O channel check error has occurred
5		Output from timer channel 2
4		Refresh detect; toggles once per refresh cycle
3		I/O channel check NMI latch (See Figure D-2)
	1	Disabled
	0	I/O channel check is enabled
2		SPU RAM parity error latch (See Figure D-2)
	1	Disabled
	0	Parity error is enabled
1	1	Speaker data from timer channel 2 is enabled to drive speaker circuit.
0	1	Gate to timer channel 2 is enabled
		When the CPU writes port 61H:
Bit	**Data**	**SPU Control Port Description**
7-4		Not used
3	1	Disable and clear I/O channel check.
2	1	Disable and clear on-board parity NMI
1	1	Enable the data from timer channel 2 to drive speaker circuit.
0	1	Enable gate to timer channel 2. (speaker data)

Speaker Control

Figure D-1 shows the relationship of the timer channel 2 and the rest of the speaker circuit.

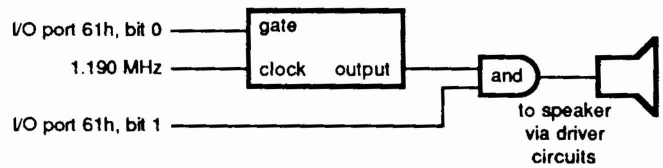


Figure D-1. Speaker Control Circuit

The sound-related EX-BIOS functions are the recommended method of accessing the speaker functions (see Chapter 8).

Keyboard I/O Ports

Keyboard Command Port (64H): This port is Used to write commands to the 8042 keyboard controller.

Keyboard Extended Command Port (68H): This port provides a data/command path to the 8042 not conflicting with the industry standard I/O ports 60H and 64H.

HP-HIL Controller (6CH through 6FH): This set of I/O ports provides the communication mechanism to the HP-HIL controller.

Real-Time Clock Ports

Real-Time Clock and CMOS RAM ports 70H and 71H provide the interface to the MC146818 real-time clock (RTC). See Appendix C for further details.

Hard Reset Enable Port

Hard Reset Enable Register (Port 78H): This write-only port enables the hard reset line from the HP-HIL controller. Table D-9 shows the bit definitions for this port.

Table D-9. Hard Reset Enable Register

Bit	Data	Description
7	1	Enable hard reset from HP-HIL controller chip.
	0	Disable and clear hard reset from HP-HIL controller chip.
6-0		Reserved.

NMI Sources and Involved I/O Ports

The non-maskable interrupt (NMI) of the CPU is attached to circuitry which allows multiple sources to cause an NMI. Each of these sources can be disabled individually as well as collectively.

Figure D-2 is a block diagram of the NMI circuit.

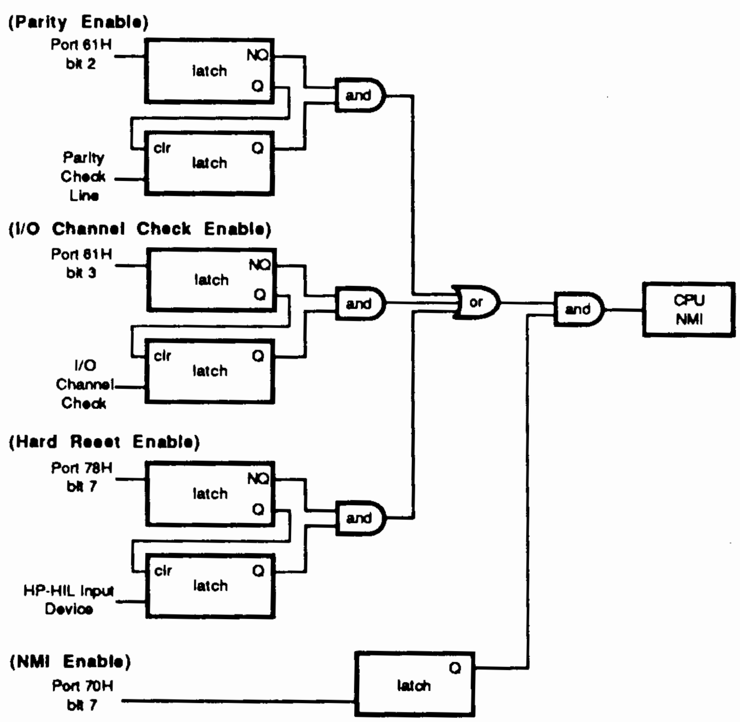


Figure D-2. NMI Circuit Block Diagram

E

Default Device Mapping

Table E-1 describes the device mappings which are set up during SYSGEN. The default mapping device is listed first. Other mappable devices are listed following the default device.

Table E-1. Input System

Physical Device	Logical Device	Mappable Driver
Mouse	Cursor Control Pad	V_PGID_CCP V_LPOINTER V_LTOUCH V_LTABLET
Rotary Knob	Cursor Control Pad	V_PGID_CCP V_LPOINTER V_LTOUCH V_LTABLET
Touchscreen	Touchscreen	V_LTOUCH V_LPOINTER V_PGID_CCP V_LTABLET
Tablet	Cursor Control Pad	V_LTABLET V_LPOINTER V_LTOUCH V_PGID_CCP
Vectra Keyboard/DIN	Keyboard Subsystem	V_8042
Compatibility Function keys	V_FUNCTION	non-mappable
HP Function keys	V_SOFTKEY	SKEY2FKEY V_OFF V_RAW

Table E-1. Input System (Cont.)

Physical Device	Logical Device	Mappable Driver
Typewriter Keypad	V_QWERTY	non-mappable
Numeric Keypad	V_NUMPAD	non-mappable
Cursor Control Keypad	V_CCP	V_CCPCUR V_OFF V_RAW V_CCPGID (if installed)

Discs

DISC A:	Flexible Disc 0 Upper Drive
DISC B:	Flexible Disc 1 Lower Drive
DISC C:	Internal Hard Disc
DISC D:	External Disc
DISC E:	RAM disc

Discs on the system are only mappable using the MS-DOS ASSIGN command.

Character I/O Devices

LPTI:	OR PRN:
COM1:	Serial Port 0
COM2:	Serial Port 1
COM3:	Serial Port 2
COM4:	Serial Port 3
LPT1:	or PRN: Parallel Port 0
LPT2:	Parallel Port 1
LPT3:	Parallel Port 2

These ports are only mappable using the MS-DOS MODE command.

F

Driver Writer's Guide

This appendix describes how a programmer can add drivers to the ROM BIOS. One of the important features of the EX-BIOS is the ease with which it can be expanded. This capability allows programmers to take full advantage of HP system components (such as the HP-HIL touchscreen, mouse, tablet, etc.). In addition, the EX-BIOS architecture provides a simple, yet powerful way to integrate OEM and third-party products into the system. This appendix is intended for all programmers and advanced users who wish to utilize EX-BIOS capabilities not supported by system software. It assumes that the reader is familiar with the contents of Chapters 1 through 8, iAPX286 or iAPX386 programming, DOS concepts in general, and DOS installable device drivers in particular. The reader should consult the publications listed under the References section at the end of this manual for additional information on these topics.

Introduction

This appendix presents two examples of how drivers that interface to the system's EX-BIOS can be written. All aspects of how a driver gets connected and used through the EX-BIOS are discussed.

The typical steps involved in connecting a driver into the EX-BIOS are:

- A driver added to the system can be one of three types: ROM driver, MS-DOS installable device driver or MS-DOS command that executes and stays resident.
- The driver gets called to initialize. At this point the driver will determine what machine it is executing on, obtain memory for its data segment, get an EX-BIOS vector address assigned and be added to the HP_VECTOR_TABLE.
- Any time after initialization, the driver can respond to service requests in two ways. It responds to a hardware service request when it is called with its F_ISR (AH = 0) function or it responds to an application service request when it is called with any other driver-specific function.

The above sequence is a general description of a driver's life cycle. It is not necessary that all drivers follow the same steps. The sections below outline what are the necessary elements of an EX-BIOS driver.

Note: For a detailed explanation of the calls to V_SYSTEM (i.e., INT V_SYSTEM) used below see Chapter 8.

Installation of Device Drivers

Each type of device driver is installed in a different manner depending on how it is brought into the system. There are three ways that an EX-BIOS driver can be installed in the system. An I/O adapter card can have an EX-BIOS driver which can be installed in the system when the adapter's ROM gets called to initialize during the SYSGEN process. The adapter's initialization routines can use all of the V_SYSTEM functions to properly connect the driver. Note that because the adapter's code modules are initialized during the system generation process (SYSGEN), an EX-BIOS driver on an adapter card can not depend on other EX-BIOS drivers already being present and initialized (V_SYSTEM is the only driver usable at this point).

An MS-DOS installable device driver can also install an EX-BIOS driver. The driver must have two interfaces, one driver interface for MS-DOS and one driver interface for the EX-BIOS functions. This type of EX-BIOS driver can use all other EX-BIOS drivers already present in the system.

Finally an MS-DOS command that stays resident can also be used to install an EX-BIOS driver. This driver can use all previously installed EX-BIOS drivers. This is the preferred method of installing EX-BIOS drivers since it only requires the EX-BIOS driver interface and functions.

Initialization

This section covers the possible steps the driver must take to insure proper initialization.

Product Identification

This section discusses several methods available through ROM BIOS functions for software to determine whether its host is an HP Vectra.

HP Vectra Feature/Revision Identification (V_SCOPY):

The V_SCOPY (00H) vector entry has a data segment (DS) that points to the system's copyright string. The driver can look at this string to determine if the machine is an HP Vectra. The following example illustrates how to get this string:

```
MOV  BP, V_SCOPY   ; Call the COPYRIGHT
PUSH DS            ; vector which will set
CALL SYSCALL       ; the DS and return
PUSH DS            ; Save DS of copyright
                   ; string in ES.
POP  ES            ; ES:0 is address
                   ; of string
POP  DS            ; Recover old DS.
```

HP Vectra Indicator Word, Revision Word, and Date Codes

At ROM address 0F00F8H, the HP Vectra series of personal computers have the following data.

```
0F000:00F8  DB  'HP'
0F000:00FA  Product ID byte  (Bits 7-5 = CPU, Bits 4-0 = SPU)
0F000:00FB  Machine Capability Marker
0F000:00FC  Version Number
0F000:00FE  Date Code

For a complete definition of product description code, refer to
the "Product Identification" section in Appendix B.
```

This code can be used to discern the HP Vectra series of personal computers from other industry-standard products and thus take advantage of the unique features of the HP Vectra series of personal computers. (However, this method is not the preferred method.)

STD-BIOS Extended Functions

The STD-BIOS Extended Functions Fnn_INQUIRE (6F00H) indicate to the calling application that STD-BIOS extended functions are loaded and have not been replaced. The STD-BIOS drivers listed in Table F-1 below support this function.

Table F-1. STD-BIOS Drivers that Support Fnn_INQUIRE

Interrupt	Function
INT 10	VIDEO
INT 14	SERIAL
INT 16	KEYBOARD
INT 17	PRINTER

To find out if the STD-BIOS extensions for the Video driver are in place, use the following code:

```
    MOV   AX, F10_INQUIRE     ; Call video
                              ; function (6F00)
    MOV   BX, OFFFFH          ; Make sure
                              ; BX <> 'HP'
    INT   INT_VIDEO           ; Interrupt 10H
    CMP   BX, 'HP'            ; Are video
                              ; extensions
                              ; present?
    JE    VIDEO_EXTENSIONS_PRESENT
VIDEO_EXTENSIONS_NOT_PRESENT:

VIDEO_EXTENSIONS_PRESENT:
```

Obtaining Memory From the EX-BIOS

The system allows EX-BIOS drivers to obtain limited amounts of memory independent of the operating system. This feature is especially important for I/O ROM adapters since their cost can be reduced if they do not require dedicated RAM. When the I/O ROM module is initialized, it can ask for EX-BIOS memory.

This feature of the EX-BIOS system can also be utilized by application programs and system software. Any program needing a limited amount of RAM outside the operating system domain can obtain this from the EX-BIOS system.

The functions F_RAM_GET and F_RAM_RET in the V_SYSTEM driver can be used to manipulate the EX-BIOS free memory. The driver can also use the installation functions F_INS_FREEGETDS or F_INS_FIXGETDS to obtain free memory. See Chapter 8 for more details of these functions.

Getting a Free Vector

In order for an application to access an EX-BIOS driver it must call the driver through the HP_VECTOR_TABLE. Thus, each driver must request a free vector from this table.

To get a free vector from the HP_VECTOR_TABLE, a driver can use the function F_INS_XCHGFREE, F_INS_FREEOWNDS, F_INS_FREEGETDS or F_INS_FREEGLBDS in the V_SYSTEM driver. Each of these functions installs the driver at the next available free vector. (See Table 9-3 "V_SYSTEM Driver Function Code Summary" for the numerical values of the above functions.

Once the driver has a vector address installed in the table, an application can call the driver by loading BP with the vector address of the driver and doing a CALL SYSCALL.

EX-BIOS Driver Functions

EX-BIOS drivers support a standard set of functions and subfunctions. Nine standard function codes are defined, and several of these functions have subfunctions defined within them. These functions and subfunctions are summarized in Table F-2. A detailed description of each defined function and subfunction follows.

If a driver receives a function it does not implement, it must return a status code of RS_UNSUPPORTED (02H) in the AH register. This lets the application know that the driver has not handled this function, but that it can continue if it is appropriate. This protocol frees the driver from having to implement all the defined functions and allows applications to call drivers in a consistent way.

If a driver receives a function code that it does not implement, it may also "delegate" the function to another driver. A driver may be written so that it calls another driver when it receives an unimplemented function or subfunction request.

Programmers may write drivers that implement functions and subfunctions that are not defined. However, two guidelines should be observed when defining additional functions or subfunctions. First, whenever possible, newly defined function or subfunction numbers should not conflict with existing numbers. Secondly, function and subfunction numbers should be consistent between drivers of the same class.

Table F-2. EX-BIOS Driver Function Code Summary

Register AH AL	Function (AH)/ Subfunction (AL)	Definition
00	F_ISR	Responds to a logical Interrupt Service Request (ISR).
	F_SYSTEM	Executes one of several standard subfunctions.
02 00	SF_INIT*	Starts the initialization of a driver.
02 02	SF_START*	Completes the initialization process of the driver.
02 04	SF_REPORT _STATE	Reports the state of the driver.
02 06	SF_VERSION _DESC*	Reports the revision number and datecode of the driver.
02 08	SF_DEF _ATTR	Reports the default configuration of the driver.
02 0A	SF_GET _ATTR	Reports the current configuration of the driver.
02 0C	SF_SET _ATTR	Overrides the current configuration of the driver.
02 0E	SF_OPEN	Reserves the driver for exclusive access. Requests any resources required by the driver.
02 10	SF_CLOSE	Releases the driver from exclusive access.
02 12	SF_TIMEOUT	Reports to the driver that a requested timeout has occurred.
02 14	SF_INTERVAL	Reports to the driver that a requested 60 Hz interval has expired.
02 16	SF_TEST	Performs a hardware test.
	F_IO_CONTROL	Executes the following subfunctions and any driver dependant subfunctions.
04 00	SF_LOCK	Reserves the sub-address device specified for exclusive access.
04 02	SF_UNLOCK	Releases the sub-address specified from the exclusive access.
06	F_PUT_BYTE	Writes a byte of data.
08	F_GET_BYTE	Reads a byte of data.
0A	F_PUT_BUFFER	Writes a variable length buffer of data (supported by character devices).
0A	F_PUT_BLOCK	Writes a fixed length buffer of data (supported by block devices).
0C	F_GET_BUFFER	Reads a variable length buffer of data (supported by character devices).
0C	F_GET_BLOCK	Reads a fixed length block of data (supported by block devices).
0E	F_PUT_WORD	Writes a word of data.
10	F_GET_WORD	Reads a word of data.

Note: Functions marked with an asterisk (*) should be supported by all drivers. These functions may perform no useful function. However, they should return a status code of RS_DONE (06H) or RS_SUCCESSFUL (0) as opposed to RS_UNSUPPORTED (02H).

The following is a list of predefined driver function codes and a brief description of their purpose and parameters:

EX-BIOS Driver Function Definitions

F_ISR (AH = 00H)

This function processes either a logical or a physical interrupt event. It reports whether or not it handled the event through its Return Status Code (see Table F-2). The driver may require the service of its parent driver to handle the interrupt.

EX-BIOS drivers do not usually enable interrupts (STI) while processing this function code. Drivers should service this interrupt within 250 microseconds or maintain interrupts off for no more than 250 microseconds at a time. Drivers should expect 40 bytes of stack when called. If a driver enables interrupts it must provide 40 bytes of stack for other ISR's.

```
On Entry: AH = F_ISR (00H)

On Exit:  AH = RS_SUCCESSFUL (00H)
          or RS_NOT_SERVICED (04H)
```

F_SYSTEM (AH = 02H)

This function contains a set of subfunctions that execute system-oriented tasks. These subfunctions include driver setup, configuration, and control. The F_SYSTEM subfunctions are described in detail below.

SF_INIT (AX = 0200H)

This starts the initialization process of a driver. The function does not return to the caller until the driver is ready to be called by another driver. All system services (V_SYSTEM) are assumed to be operational when a driver is called by this function.

The driver is responsible for a brief hardware check and for reporting RS_FAIL if the test failed. A driver need only execute a test procedure if it directly interfaces to physical hardware.

If the driver requires EX-BIOS RAM, the BX and DX registers can be used to reserve available memory (see Chapter 8).

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INIT (00H)
          BX = "last used DS"
          BP = Driver's vector address

On Exit:  AH = Return Status Code
          BX = New "last used DS"
```

Recommended for hardware test failure:

```
     AH = RS_FAIL (02H)
  ES:DI = pointer to a string of information
          about the nature of the error
     CX = length of the string pointed to
          by ES:DI
```

SF_START (AX = 0202H)

This function notifies a driver that it may call other drivers for any additional setup it may require. All other ROM drivers and ROM services are present, active and capable of being accessed. This function does not usually return to the caller until all its internal and external setup is complete.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_START (02H)
          BP = Driver's vector address

On Exit: AH = Return Status Code
```

SF_REPORT_STATE (AX = 0204H)

Reports a word of status or state information to the caller in the DX register. The format of the state information will be presented bit wise and should be presented in the same format for all drivers of the same class.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_REPORT_STATE (04H)
          BP = Driver's vector address

On Exit: AH = Return Status Code
         BX = State of Driver
```


SF_VERSION_DESC (AX = 0206H)

Reports the version number of the driver code and an optional describe record which contains other driver-dependent information.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_VERSION_DESC (06H)
          BP = Driver's vector address

On Exit: AH = Return Status Code
         BX = Version number,
              YYWW is a BCD number where,
                WW is the week of the year
                YY is the number of years
                      since 1960
         CX = Number of bytes in data buffer
      ES:DI = Pointer to describe record
```

SF_DEF_ATTR (AX = 0208H)

Returns a pointer in ES:DI to a parameter block containing the driver's default configuration values. This function does not set the defaults; it only reports them.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_DEF_ATTR (08H)
          BP = Driver's vector address

On Exit: AH = Return Status Code
         CX = Number of bytes in data buffer
      ES:DI = Pointer to a data buffer
```

SF_GET_ATTR (AX = 020AH)

Reports the configuration values defined by the parameter block. Baud rates, HP-IB addresses, etc. may be reported by this command.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_GET_ATTR (0AH)
          BP = Driver's vector address

On Exit: AH = Return Status Code
         CX = Number of bytes in data buffer
      ES:DI = Pointer to a data buffer
```

SF_SET_ATTR (AX = 020CH)

Sets the parameter block defined by ES:DI as the configuration values. Baud rates, HP-IB addresses, etc. may be defined by this command.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_SET_ATTR (0CH)
          BP = Driver's vector address
          CX = Number of bytes in data buffer
       ES:DI = Pointer to a data buffer

On Exit: AH = Return Status Code
      ES:DI = Pointer to a data buffer
```

SF_OPEN (AX = 020EH)

Allows exclusive access to this driver. All resources required for driver operation will be acquired at this time. This function has special meaning for the the HP-HIL driver, the HP-IB driver and the HP-IL driver. Since these drivers support shared interfaces, control of the resource HP-HIL (obtained from the driver V_HPHIL), control of the HP-IB (in contention with other PCs on the bus), and control of the HP-IL (in contention with other PCs on the loop) is requested and obtained. Control should be kept until a single operation is performed on the resource. A status of RS_BUSY will be reported if the device has previously been opened. RS_SUCCESSFUL will be reported if the device is available. A busy status does not prevent access to the driver. All functions will execute (perhaps improperly) whether a driver has been opened or not.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_OPEN (0EH)
          BP = Driver's vector address

On Exit: AH = Return Status Code
```

SF_CLOSE (AX = 0210H)

Closes the requested resource. Again, this function has special meaning for the interface class of devices, HP-IB, HP-HIL, and HP-IL. The driver goes to a state where control can be obtained by or passed to another controller.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_CLOSE (10H)
          BP = Driver's vector address

On Exit: AH = Return Status Code
```

SF_TIMEOUT (AX = 0212H)

Reports to the driver that its timer event number has occurred.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_TIMEOUT (12H)
          BP = Driver's vector address

On Exit: AH = Return Status Code
```

SF_INTERVAL (AX = 0214H)

Reports to the driver that its interval event number has occurred.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_INTERVAL (14H)
          BP = Driver's vector address

On Exit: AH = Return Status Code
```

SF_TEST (AX = 0216H)

The driver performs a hardware test and reports RS_FAIL if the test failed. A driver need only execute a test procedure if it directly interfaces to physical hardware.

```
On Entry: AH = F_SYSTEM (02H)
          AL = SF_TEST (16H)
          BP = Driver's vector address

On Exit: AH = Return Status Code

         On test failure:

         CX = The length of the string pointed
              to by ES:DI
      ES:DI = Pointer to a string of
              information about the nature of
              the error
```

F_IO_CONTROL (AH = 04H)

This is a collection of driver-dependant control subfunctions. Drivers of the same class should implement similar subfunctions. The following is a list of predefined driver subfunction codes and a brief description of their purpose and parameters:

SF_LOCK (AX = 0400H)

Reserves the indicated addresses on an already allocated driver for exclusive access.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_LOCK (00H)
       DH,DL = Major and minor address
               (Optional)
          BP = Driver's vector address

On Exit: AH = Return Status Code
```

SF_UNLOCK (AX = 0402H)

Releases the indicated address from exclusive access.

```
On Entry: AH = F_IO_CONTROL (04H)
          AL = SF_UNLOCK (02H)
       DH,DL = Major and minor address
               (optional)
          BP = Driver's vector address

On Exit: AH = Return Status Code
```

F_PUT_BYTE (AH = 06H)

This is a generic put data byte function.

```
On Entry: AH = F_PUT_BYTE (06H)
          AL = Data byte
          BP = Driver's vector address

On Exit: AH = Return Status Code
```

F_GET_BYTE (AH = 08H)

This is a generic get data byte function.

```
On Entry: AH = F_GET_BYTE (08H)
          BP = Driver's vector address

On Exit: AH = Return Status Code
         AL = Data byte
```

F_PUT_BUFFER OR F_PUT_BLOCK (AH = 0AH)

Puts a number of bytes to a device. The difference between a buffer device and a block device is that a buffer device accepts variable length records, while a block device accepts fixed length records. Thus, a printer is a data buffer device and a disc is a block device. Usually, a block device requires more parameters than a data buffer device; consequently, there is a different format for parameter passing.

F_PUT_BUFFER (AH = 0AH)

This is a generic put data buffer or put data block function. Either a string write or a disc block write could use this function.

```
On Entry: AH = F_PUT_BUFFER (0AH)
          CX = Data byte count or block count
       ES:DI = Pointer to data buffer
          BP = Driver's vector address

On Exit: AH = Return Status Code
```

F_PUT_BLOCK (AH = 0AH)

Writes a fixed block of data to a block device.

```
On Entry: AH = F_PUT_BLOCK (0AH)
          DH = Major number
          DL = Minor number
       ES:DI = Command Block

               Word 0,1: Data transfer address
               Word 0,2: Block count
               Word 0,3: Block address LSW
               Word 0,4: Block address MSW
               (for some devices this word
               is ignored).

          BP = Driver's vector address

On Exit: AH = Return Status Code
         BX = Operation status
```

F_GET_BUFFER OR F_GET_BLOCK (AH = 0CH)

F_GET_BUFFER (AH = 0CH)

This is a generic get buffer or get block function. Either string reads or disc block reads could use this function.

```
On Entry: AH = F_GET_BUFFER (0CH)
          CX = Byte count or block count
       DS:SI = Pointer to data buffer
          BP = Driver's vector address

On Exit: AH = Return Status Code
```

F_GET_BLOCK (AH = 0CH)

Reads a fixed length block of data from a device.

```
On Entry: AH = F_GET_BLOCK (0CH)
          DH = Major number
          DL = Minor number
       ES:DI = Command Block

               Word 0,1: Data transfer address
               Word 0,2: Block count
               Word 0,3: Block address LSW
               Word 0,4: Block address MSW
               (For some devices this word
               is ignored).

          BP = Driver's vector address

On Exit: AH = Return Status Code
         BX = Operation status
```

F_PUT_WORD (AH = 0EH)

This is a generic put word of data function. If the destination device is byte wide, then the byte in the DL register is written first, followed by the byte in the DH register.

```
On Entry: AH = F_PUT_WORD (0EH)
          DX = Data word
          BP = Driver's vector address

On Exit: AH = Return Status Code
```

F_GET_WORD (AH = 10H)

This is a generic get word of data function. If the source device is byte wide, then the first byte is read into the DL register, and the second byte is read into the DH register.

```
On Entry: AH = F_GET_WORD (10H)
          BP = Driver's vector address

On Exit: AH = Return Status Code
         DX = Data word
```

Return Status Codes

The conventions for assigning return status codes are as follows:

If possible, use a return status that has already been defined.

Error conditions should be reported with a negative byte (0FEH--080H).

Status or exceptional conditions "soft errors" should be reported with a positive byte (02--7EH).

Good Status is always reported as 00H.

Table F-3 summarizes the already assigned status codes.

Table F-3. EX-BIOS Assigned Return Status Codes

Return Status	Code	Indication
000H	RS_SUCCESSFUL	The requested function executed correctly.
002H	RS_UNSUPPORTED	The requested function or subfunction is not implemented or is unsupported.
004H	RS_NOT_SERVICED	The requested function was not executed by this driver. Any drivers which are chained on this interrupt vector should be called in turn until a return status of RS_SUCCESSFUL or some other error is reported.
006H	RS_DONE	This return status is used by the Input System translators to indicate that an ISR event has been handled and no further processing should be done.
0FEH (-02H)	RS_FAIL	The driver failed the operation in an error state.
0FCH (-04H)	RS_TIMEOUT	The device timed out on a physical event in an error state.
0FAH (-06H)	RS_BAD _PARAMETER	The driver received a bad parameter.
0F8H (-08H)	RS_BUSY	The requested driver is busy.
0F6H (-0AH)	RS_NO_VECTOR	HP_VECTOR_TABLE is out of RAM or room for more drivers.
0F4H (-0CH)	RS_OFFLINE	Device is offline.
0F2H (-0EH)	RS_OUT_OF _PAPER	Device is out of paper.

Driver Headers

The EX-BIOS driver header (HP_SHEADER) is a formatted data structure similar to the DOS device driver's header. It defines the attributes of a driver, defines the linkage of a driver and identifies the driver. It also allows the programmer to define how the driver links with other drivers.

All EX-BIOS drivers must have an HP_SHEADER. Programmers are not required to provide a complete HP_SHEADER to use the HP_VECTOR_TABLE. But, if they choose to take advantage of the advanced features of the EX-BIOS the programmer must implement a complete HP_SHEADER. Table F-4 shows a complete driver header and what fields must be present.

Table F-4. Driver Header Table

Offset	Type	Variable	Definition
0	Word	DH_ATR*	Driver Attribute Field
2	Word	DH_NAME_INDEX	Driver String Index Field
4	Word	DH_V_DEFAULT	Driver's Default Logical Device Vector
6	Word	DH_P_CLASS**	Driver's Parent Class
8	Word	DH_C_CLASS**	Driver's Child Class
0AH	Word	DH_V_PARENT**	Driver's Parent Vector
0CH	Word	DH_V_CHILD**	Driver's Child Vector
0EH	Byte	DH_MAJOR**	Subaddress Field
0FH	Byte	DH_MINOR**	Subaddress Field

*This is the only field required for a driver to be in the HP_VECTOR_TABLE.
**These fields are only required by drivers that want to do device mapping.

HP_SHEADER Fields

DH_ATR: Each bit in the DH_ATR field indicates a property of the driver for device mapping purposes. These bits are defined in Table F-5.

Table F-5. Device Attributes Bits

Bit	Data	ATR Name	Description
15	1	ATR_HP	The following bytes form a complete driver header.
	0		The bytes that follow are not a driver header.
14		ATR_DEVCFG	Reserved.
13	1	ATR_ISR	The driver can be mapped with DH_V_PARENT.

Table F-5. Device Attributes Bits (Cont.)

Bit	Data	ATR Name	Description
12	1	ATR_ENTRY	The driver can be mapped with DH_V_CHILD.
11-9		ATR_TYPE _MASK	These three bits indicate the driver type.
	000	ATR_RSVD	This is a reserved vector.
	001	ATR_FREE	This is a free vector. The V_SYSTEM service allocates free vectors to new drivers upon request.
	010	ATR_SRVC	This driver is an EX-BIOS service.
	011	ATR_LOG	This is a logical driver. Its mapping direction is from parent to child.
	100	ATR_IND	This is a mappable driver that cannot be the last in the chain of drivers.
	101	ATR_BOT	This is a mappable driver that is the last in a chain of drivers. This driver can only be a child driver. This driver maps with ATR_LOG, ATR_IND and ATR_BOT drivers.
	110	ATR_INP	This driver is an input driver and is mappable.
	111		Reserved
8		ATR_STRING	Reserved
7	1	ATR_MAP _CALL	This driver's SF_START subfunction should be called whenever the driver is remapped.
6,5		ATR_SUBADD	These bits specify what type of major and minor addresses the driver requires.
	00	ATR_NOADDR	The driver does not require any address.
	01	ATR_MAJOR	This driver requires that a major address be specified and stored in the parent driver's DH_MAJOR header record. The range of possible major addresses is 0 through the contents of this header's DH_MAJOR.

Table F-5. Device Attributes Bits (Cont.)

Bit	Data	ATR Name	Description
	10	ATR_MINOR	This driver requires that a minor address be specified and stored in the parent driver's DH_MINOR header record. The range of possible MINOR addresses is 0 through the contents of this header's DH_MINOR. A driver cannot require a minor address unless it also requires a major address.
	11	ATR_MID	This driver requires a major address, a minor address, and a mid address. The minor address field is split into an upper and a lower nibble, with the upper nibble indicating the mid address and the lower nibble indicating the minor address. The range of addresses possible is specified by the child physical driver.
4	0	ATR_PSHARE	This driver cannot be shared between several parent drivers.
3	0	ATR_CSHARE	This driver cannot be shared between several child drivers.
2	1	ATR_ROM	This driver header is in ROM and cannot be altered unless copied to RAM. 1 Reserved
1		ATR_YIELD	Reserved.
0		Reserved	

DH_NAME_INDEX: The DH_NAME_INDEX is used to derive the localization string index of the driver. This is given by the function F_STR_GET_STRING in the V_SYSTEM driver. See Chapter 8 for additional information.

DH_V_DEFAULT: The DH_V_DEFAULT field contains the driver's default vector address.

DH_P_CLASS and DH_C_CLASS: In conjunction, these fields indicate which drivers may be mapped together. DH_P_CLASS and DH_C_CLASS are bit masks. Each bit position represents a set of drivers. If a bit is set then the driver is in that set of drivers. The DH_P_CLASS field indicates a driver is in from 0 to 16 different driver sets. A driver can only map to another driver if its DH_P_CLASS field matches at least one bit position of another driver's DH_C_CLASS field. Furthermore, DH_ATR field is another condition of mapping. The bits are defined in Table F-6.

Table F-6. Class Bit Positions

Hex	Bit	Class Name	Definition
8000	0FH	CL_KBDFC	This set of drivers maps to the f1 through f8 softkeys of the keyboard.
4000	0EH	CL_KBD	Keyboard (this is not the device accessed through INT 16).
2000	0DH	CL_CCP	Cursor pad device (for example, V_CCPCUR, V_CCP NUM, V_OFF, V_RAW, V_CCP, V_FUNCTION).
1000	0CH	CL_CON	This set of devices map to the console device.
0800	0BH	CL_BYTE	Serial output device, which may be capable of limited input.
0400	0AH	CL_COMM	Reserved
0200	09H	CL_INTERFACE	An interface class controlling multiple resources transparent to the operating system. It provides major, middle, and minor address modes for the calling application or driver. Examples are the HP-HIL driver, the HPIB driver, and the HPIL driver.
0100	08H	CL_FILT	Serial output device filter. This driver can be mapped in between a logical driver and a physical driver and it can translate from one character set to another.
0080	07H	CL_BLK	Addressed block device.
0040	06H	CL_BOOT	Logical device used as the priority boot device. If set on a physical device, the device is capable of being a boot device. Typically a physical driver would have both the CL_BOOT bit set and the CL_BLK bit set.

Table F-6. Class Bit Positions (Cont.)

Hex	Bit	Class Name	Definition
0020	05H	CL_LGID	Logical graphics input device (for example V_LTABLET, V_LPOINTER, physical GID devices and the keyboard driver). This class maps to logical devices which are not the child of another driver.
0010	04H	CL_PGID	This class of driver can map to a device which is the child of another driver.
0008	03H	CL_GID	This class is reserved for all drivers which can map to an event.
0004	02H	CL_PTS	Physical touch device (for example, physical GID drivers or V_LTOUCH).
0002	01H	CL_01	Reserved
0001	00H	CL_00	Class Extension Bit
			Special Group Classes
FFFF	-	CL_ALL	This device maps to all other devices (V_PNULL).
0000	-	CL_NULL	This device maps to no other driver.

DH_V_PARENT: The DH_V_PARENT field contains a vector to the driver that is called when the current driver receives an F_ISR function code that it cannot or doesn't know how to process.

DH_V_CHILD: The DH_V_CHILD field contains a vector to the driver that is called if this driver decides it cannot handle the request function (as long as that function is not F_ISR).

DH_MAJOR: Major address range.

DH_MINOR: Minor address range.

See the HP_SHEADER macro definition in the equate files listed in Appendix E.

Driver Mapping

Two drivers may be mapped together if the drivers have matching parent and child class records. The mapping rule for the drivers is defined in Table F-7.

Table F-7 PARENT/CHILD Driver Mapping Rules

Parent E I	Child E I	Connection Rule
0 0	0 0	Drivers are not to be connected
0 0	0 1	"
0 0	1 0	"
0 0	1 1	"
0 1	0 0	"
0 1	0 1	Child's DH_V_PARENT <-- parent's vector address
0 1	1 0	Drivers can not be connected
0 1	1 1	Child's DH_V_PARENT <-- parent's vector address
1 0	0 0	Drivers are not connected
1 0	0 1	"
1 0	1 0	Parent's DH_V_CHILD <-- child's vector address
1 0	1 1	Parent's DH_V_CHILD <-- child's vector address
1 1	0 0	Drivers are not connected
1 1	0 1	Child's DH_V_PARENT <-- parent's vector address
1 1	1 0	Parent's DH_V_CHILD <-- child's vector address
1 1	1 1	Child's DH_V_PARENT <-- parent's vector address and Parent's DH_V_CHILD <-- child's vector address

Where,
E = ATR_ENTRY bit state
I = ATR_ISR bit state

Accessing Driver from an Application

When an application needs to access a driver, the following sequence must take place:

```
MOV BP, driver's vector address     ; i.e.
                                    ; V_SYSTEM (12H)
MOV AH, function code
MOV AL, subfunction code
.
.                                   ; any other data passed
.                                   ; in registers
PUSH DS                             ; Saves application's DS
CALL SYSCALL
POP DS
```

Examples of EX-BIOS Drivers

NOTE

Since the HP interrupt number can change, all "int HP_ENTRY" lines in the following examples should be replaced with "CALL SYSCALL" (this routine finds the current HP interrupt number).

Cursor Pad Scancode To HP Mouse Driver

The first example driver is called CPP2GID. This driver implements the V_CCPGID EX-BIOS driver. As such, it translates from cursor control pad keys into graphics input device data.

The driver is installed into the HP_VECTOR_TABLE. The SF_INIT subroutine of the driver asks for enough EX-BIOS RAM to store the driver header and describe record. The DH_V_PARENT field of the V_CCPGID driver header is initialized to the installable HP Mouse driver, V_LHPMOUSE (this driver is shipped on a separate disc with all Vectra series of personal computers). The DOS driver portion calls SF_START of the EX-BIOS driver. SF_START initializes the DH_V_PARENT field of the V_CCP driver header to V_CCPGID. Then the installable V_LHPMOUSE driver is called with the override function.

The installable driver completes initialization by printing an initialization completed message and returning to DOS.

Now when the keyboard driver calls V_CCP to process a cursor control pad key, V_CCP calls V_CCPGID. The F_ISR of V_CCPGID decodes which key was actually pressed. The driver converts the cursor movement keys (up, down, left, and right) into relative movement data. If the key pressed was an insert or delete key, it is reported as the left or right button respectively. The driver first changes the describe record and then reports either a button press or a button release. After the input data is given to installable V_LHPMOUSE, the data is available thru the INT 33H STD-BIOS driver.

NOTE

As mentioned before, the HP_ENTRY interrupt number is defaulted at 006FH - but this number can *change*. The following examples show HP_ENTRY at its default, but when accessing EX-BIOS drivers you should use a "CALL SYSCALL" in place of "int HP_ENTRY."

CCP__TO__GID__FILTER

```
                                        286c
                                        page 59,132
                                        title CCP_TO_GID_FILTER installable driver
                                        ;***DRIVER HEADER*************************************************
                                        ;
                                        ; NAME  CCP_TO_GID_FILTER Installed DRIVER
                                        ;
                                        ; DESCRIPTION:  This is an EX-BIOS driver which converts cursor
                                        ;               control pad cursor keys into GID, T_REL16, movements
                                        ;               It is a brother to the V_CCPCUR, V_CCPNUM, V_RAW, and
                                        ;               V_OFF, filters of the V_CCP translator
                                        ;
                                        ;               One cursor key report generates one micky in the direction
                                        ;               indicated by the cursor pad key.  In addition the cursor
                                        ;               control pad <Ins> key is mapped to the B1 <o> mouse button
                                        ;               and the cursor control pad <DEL> key is mapped to the B2 <oo>
                                        ;               mouse button
                                        ;
                                        ;  OPERATION
                                        ;               This driver is installed through the MS-DOS installed device
                                        ;               driver system with the command line
                                        ;
                                        ;               device=CCP2GID.EXE
                                        ;
                                        ;               The driver links itself into the HP_VECTOR_TABLE and maps
                                        ;               itself to be the parent driver of the V_CCP driver.
                                        ;
                                        ;               The driver then returns to DOS releasing the initialization
                                        ;               code it no longer requires back to DOS.
                                        ;
                                        ;  PARAMETERS
                                        ;
                                        ;    ON ENTRY           in MS-DOS portion  es:bx points to
                                        ;                                          System Request Header
                                        ;                       in HP portion      ah contains function
                                        ;                                          code, al usually contains
                                        ;                                          the output character
                                        ;
                                        ;    ON EXIT            in MS-DOS portion  status is returned in
                                        ;                                          System Request Header
                                        ;                       in HP portion      ah contains tha return
                                        ;                                          status code
                                        ;
                                        ; REGISTERS ALTERED     in MS-DOS portion  none
                                        ;                       in HP portion      ax, bx, di, bp
                                        ;
                                        ;*****************************************************************
                                        HP_SHEADER              struc
 0000  0000                             DH_ATR                  dw      0
 0002  0000                             DH_NAME_INDEX           dw      0
 0004  0000                             DH_V_DEFAULT            dw      0
 0006  0000                             DH_P_CLASS              dw      0
 0008  0000                             DH_C_CLASS              dw      0
 000A  0000                             DH_V_PARENT             dw      0
 000C  0000                             DH_V_CHILD              dw      0
 000E  00                               DH_MAJOR                db      0
 000F  00                               DH_MINOR                db      0
 0010                                   HP_SHEADER              ends

 = 006F                                 HP_ENTRY                equ     06FH

                                        SYSCALL                 macro   vector
                                         ifnb                   <vector>
                                                                mov     bp,vector
                                         endif
                                                                int     HP_ENTRY
                                                                endm

 = 0008                                 ATR_CSHARE              equ     0008H
 = 4000                                 ATR_DEVCFG              equ     4000H
 = 8000                                 ATR_HP                  equ     8000H
 = 2000                                 ATR_ISR                 equ     2000H
 = 0600                                 ATR_LOG                 equ     0600H
 = 2000                                 CL_CCP                  equ     2000H
 = 0020                                 CL_LGID                 equ     0020H

                                        DESCRIBE                STRUC
 0000      10 [                                       db        size HP_SHEADER dup (?) ; this data is always offset by
                 ??
                     ]

 0010  ??                               D_SOURCE      db        ?       ; 7-4 (high nibble) contains the GID type
                                                                        ; 3-0 (low nibble) is the address of the device
 0011  ??                               D_HPHIL_ID    db        ?       ; device id byte returned by an HPHIL device
 0012  ??                               D_DESC_MASK   db        ?       ; describe header from HPHIL device
 0013  ??                               D_IO_MASK     db        ?       ; I/O descriptor byte from device
 0014  ??                               D_XDESC_MASK  db        ?       ; extended describe byte from device
 0015  ??                               D_MAX_AXIS    db        ?       ; maximum number of exis reported
 0016  ??                               D_CLASS       db        ?       ; device class
                                                                        ; 7-4 (high nibble) contains current class
                                                                        ; 3-0 (low nibble) contain the default class
 0017  ??                               D_PROMPTS     db        ?       ; number of buttons/prompts
                                                                        ; 7-4 (high nibble) is the number of prompts
                                                                        ; 3-0 (low nibble) is the number of buttons
```

CCP__TO__GID__FILTER

```
        0018  ??                 D_RESERVED        db       ?        ; reserved for future
        0019  ??                 D_BURST_LEN       db       ?        ; maximum burst length output to a device
                                                                     ; if devices supports more than 255 bytes then
                                                                     ; 255 bytes is the default maximum
        001A  ??                 D_WR_REG          db       ?        ; number of write registers supported by a device
        001B  ??                 D_RD_REG          db       ?        ; number of read registers supported by a device
        001C  ??                 D_TRANSITION      db       ?        ; transitions reported per button
        001D  ??                 D_STATE           db       ?        ; current state of buttons
        001E  ????               D_RESOLUTION      dw       ?        ; counts / cm (m) returned by HPHIL device
        0020  ????               D_SIZE_X          dw       ?        ; Maximum count of in units of resolution
        0022  ????               D_SIZE_Y          dw       ?
        0024  ????               D_ABS_X           dw       ?        ; data reported from device
        0026  ????               D_ABS_Y           dw       ?        ; that reports absolute data
        0028  ????               D_REL_X           dw       ?        ; date reported from device
        002A  ????               D_REL_Y           dw       ?        ; that is relitive
        002C  ????               D_ACCUM_X         dw       ?        ; these are used to accumulate scaling
        002E  ????               D_ACCUM_Y         dw       ?          remainder
        0030                     DESCRIBE                   ENDS

      = 0030                     DESCRIBE_SIZE              equ      size DESCRIBE

      = 001E                     D_CCP_STATE                equ      D_STATE + 1
      =                          D_SIZE                     equ      D_SIZE_X
      =                          D_SAMPLE_ABSOLUTE          equ      D_ABS_X
      =                          D_SAMPLE_RELATIVE          equ      D_REL_X
      =                          D_REMAINDER_ACCUM          equ      D_ACCUM_X
      =                          D_BUFFER                   equ      D_SIZE_X          ; offset where buffer begins
      = 00F0                     D_CLASS_CURRENT            equ      0F0H
      = 000F                     D_CLASS_DEFAULT            equ      00FH
                                 ; The field LD SOURCE uses the following to access the defined nibbles
      = 000F                     D_ADDR_MASK                equ      00FH
      = 00F0                     D_TYPE_MASK                equ      0F0H

      = 000E                     F_INS_FIXGETDS             equ      000EH
      = 0004                     F_IO_CONTROL               equ      0004H
      = 0002                     SF_MOUSE_OVERRIDE          equ      0002H
      = 0000                     F_TSR                      equ      0000H
      = 0002                     F_SYSTEM                   equ      0002H
      = 0002                     SF_START                   equ      0002H

                                 ;**************************************************************************
                                 ; the following structures are used to access MS-DOS driver command blocks
                                 ;**************************************************************************
                                 MSD_HEADER        macro    ATT,STRATEGY_ENTRY,ISR_ENTRY,STRING
                                                   dd       -1             ;mark as last driver in list
                                                   dw       ATT
                                                   dw       STRATEGY_ENTRY
                                                   dw       ISR_ENTRY
                                                   db       STRING
                                                   db       14 dup (?)     ; Pad so it is paragraph aligned
                                                   endm

                                 MSD_REQ_HEADER    struc             ;00; structure for access to MS driver cmds
        0000  ??                 MSD_CMDLEN        db       ?        ;00; length of cmd in bytes including data @ end
        0001  ??                 MSD_UNIT          db       ?        ;01; unit number for command
        0002  ??                 MSD_CMD           db       ?        ;02; command code
        0003  ????               MSD_STATUS        dw       ?        ;03; filler with completion status before return
        0005      08 [                             db 8 dup (?)      ;  ; area reserved for DOS
                          ??
                             ]


        000D  ??                 MSD_MEDIA         db       ?        ;13; most cmds have this defined in the data area
        000E  ????               MSD_TRANS         dw       ?        ;14;
        0010  ????                                 dw       ?        ;16
        0012  ????               MSD_COUNT         dw       ?        ;18;
        0014  ????               MSD_START         dw       ?        ;20;
        0016                     MSD_REQ_HEADER             ends

                                 MSD_INIT_CMD               struc
        0000      0D [                             db       13 duo (?)    ;first cover header area
                          ??
                             ]

        000D  ??                 MSD_UNIT_COUNT    db       ?        ; 0B ;number of units service by this driver
        000E  ????               MSD_END_OFFSET    dw       ?        ; 0C ;offset of end of code
        0010  ????               MSD_END_SEG       dw       ?        ; 0E ;segment address of end of code
        0012  ????               MSD_BPB_OFFSET    dw       ?        ; 12
        0014  ????               MSD_BPB_SEG       dw       ?        ; 14 ;seg offset of BPB list for units attached
        0016  ??                 MSD_1ST_UNIT      db       ?        ; 16 ;tells driver letter of first unit
        0017                     MSD_INIT_CMD               ends

      = 0000                     MSD_INIT                   equ      0000H
      = 0003                     MSD_UNKNOWN_CMD            equ      0003H
      = 000F                     MSD_REM_MEDIA              equ      000FH
      = 0081                     MSD_ERR_STATUS             equ      10000001B         ;used as upper byte in status wrd
      = 0001                     MSD_DONE_STATUS            equ      00000001B         ;  bit 15=err bit 8=done

      = 0006                     RS_DONE                    equ      0006H
      = 0000                     RS_SUCCESSFUL              equ      0000H
      = 0002                     RS_UNSUPPORTED             equ      0002H

      = 0009                     T_KC_BUTTON                equ      0009H
      = 0041                     T_REL16                    equ      0041H
```

```
      = 0006                        V_DOLITTLE              equ     0006H
      = 00A2                        V_CCPGID                equ     00A2H
      = 00CC                        V_LHPMOUSE              equ     00CCH
      = 0012                        V_SYSTEM                equ     0012H
      = 004E                        V_CCP                   equ     004EH

      = 0080                        UP_DOWN_BIT             equ     10000000B       ;Key up or down
      = 00FF                        INIT_BUT_STATE          equ     0FFH    ;All off
      = 004C                        MSE_NUM_BUTTON          equ     004CH   ;Offset of number of button in mouse RAM
      = 0030                        CCP2GID_DESC_SIZE       equ     48
      = E608                        CCP2GID_HP_ATTR         equ     ATR_HP+ATR_DEVCFG+ATR_ISR+ATR_LOG+ATR_CSHARE

                                    ;*****************************************************************;
                                    ; MS-DOS device drivers start at an offset of 0 rather than 100h. ;
                                    ;*****************************************************************;
                                    CGROUP                  group   CODE
      0000                          CODE                    segment public 'CODE'
                                                            assume  cs:CODE, ds:NOTHING
      0000                                                  org     0
      0000                          CCP2GID_INSTALLED       LABEL   FAR
                                    ;*****************************************************************;
                                    ; This is the start of MS-DOS driver portion of the code.  It pretends ;
                                    ; to be a standard MS-DOS driver long enough to be loaded and     ;
                                    ; initialized via CONFIG.SYS.  After that this section of code will ;
                                    ; not be used.  (section 1)                                       ;
                                    ;*****************************************************************;
                                    ; This is the MS-DOS device driver header.  It must be the first thing ;
                                    ; in the code segment.  Consult the HP Vectra MS-DOS Programmers Refer- ;
                                    ; ence Manual for more  information.  BE SURE YOU DON'T RELEASE THE ;
                                    ; HEADER AREA AS AVAILABLE  MEMORY, EVEN ON AN ERROR.  THE SYSTEM WILL ;
                                    ; CRASH IF YOU DO.                                                ;
                                    ; This is the only resident portion of the DOS driver, the rest   ;
                                    ; of the DOS driver is returned to DOS memory.                    ;
                                    ;*****************************************************************;
                                            MSD_HEADER 08000h,dev_strategy,dev_int," CCP2GID"  ,device header
      0000  FF FF FF FF           +                 dd          -1                  ;mark as last driver in list
      0004  8000                  +                 dw          08000h
      0006  01AB R                +                 dw          dev_strategy
      0008  01D6 R                +                 dw          dev_int
      000A  20 43 43 50 32 47     +                 db          " CCP2GID"
      0012     0E [               +                 db          14 dup (?)      ; Pad so it is paragraph aligned.

                                     subttl CCP2GID DRIVER Main entry point
                                     page
                                    ;*****************************************************************
                                    ; CS: Relative Data Area For Driver
                                    ;*****************************************************************
      0020  ????                    sav_bx:         dw      ?
      0022  ????                    sav_cx:         dw      ?
      0024  ????                    sav_dx:         dw      ?
      0026  ????                    sav_es:         dw      ?
      0028  ????                    top_hp_entry:   dw      ?

                                    ;*****************************************************************
                                    ; This is the EX-BIOS installed driver CCP2GID.
                                    ;
                                    ;*****************************************************************
      002A                          CCP2GID_DRIVER  PROC    FAR
      002A  80 FC 00                                cmp     ah,F_ISR                ;Is the function F_ISR?
      002D  74 0B                                   je      short CCP2GID_ISR
      002F  80 FC 02                                cmp     ah,F_SYSTEM             ;Is the function F_SYSTEM?
      0032  75 03                                   jne     CCP2GID_UNSUPPORTED
      0034  E9 010D R                               jmp     CCP2GID_SYSTEM
      0037                          CCP2GID_UNSUPPORTED:
      0037  B4 02                                   mov     ah,RS_UNSUPPORTED       ;This driver doesn't support
      0039  CF                                      iret                            ;any other functions.

      003A                          CCP2GID_DRIVER  ENDP    FAR

                                                    subttl CCP2GID_isr function
                                    page
                                    ;***DRIVER HEADER*************************************************
                                    ;
                                    ;  NAME:            CCP2GID_ISR
                                    ;
                                    ;  DESCRIPTION:     This function translates valid ISR event record into
                                    ;                   mouse type movement or button reports, calls its parent driver
                                    ;                   with an ISR Event Record and then returns to the calling
                                    ;                   driver with a return status of RS_DONE
                                    ;
                                    ;  PARAMETERS
                                    ;
                                    ;    ON ENTRY:      ISR Event Record of type T_KC_HP_CCP
                                    ;                           BP = V_CCPGID
                                    ;                           DS = this drivers data segemnt
                                    ;                           AH = 0 ( F_ISR )
                                    ;
                                    ;
                                    ;    CALL PARENT:           ISR Event Record of type T_REL16 or T_KC_BUTTON
                                    ;
                                    ;       T_REL16:            AH = 0 ( F_ISR )
                                    ;                           BX = axis 0 value ( X axis or Col )
                                    ;                           CX = axis 1 value ( Y axis or Row )
```

```
                                  ;                            DH = 41H ( T_REL16 )
                                  ;                            ES 0 = describe record of V_CCPGID
                                  ;                            DL = V_CCPGID/8
                                  ;
                                  ;        T_KC_BUTTON         AH = 0 ( F_ISR )
                                  ;                            BL = 000H -  break Button 1
                                  ;                                 001H -  break Button 2
                                  ;                                 080H -  make Button 1
                                  ;                                 081H -  make Button 2
                                  ;                            DH = T_KC_BUTTON
                                  ;                            CX = 0
                                  ;                            ES 0 = this device describe record
                                  ;                            DL = V_CCPGID/8
                                  ;
                                  ;    ON EXIT
                                  ;                            AH = RS_DONE
                                  ;
                                  ; REGISTERS ALTERED          ax,  bp and ds
                                  ;
                                  ;*****************************************************************
 003A                             CCP2GID_ISR     label   near

 003A  50                                         push    ax
 003B  2E: 89 1E 0020 R                           mov     word ptr cs:sav_bx,bx   ;save the keyboard's isr
 0040  2E: 89 0E 0022 R                           mov     word ptr cs:sav_cx,cx
 0045  2E: 89 16 0024 R                           mov     word ptr cs:sav_dx,dx
 004A  2E: 8C 06 0026 R                           mov     word ptr cs:sav_es,es

 004F  8C DA                                      mov     dx,ds                   ;point to the mouse isr
 0051  8E C2                                      mov     es,dx

 0053  32 FF                                      xor     bh,bh                   ;translate the scancode to GID
 0055  83 FB 80                                   cmp     bx,80H                  ;check for cursor up
 0058  74 21                                      je      short ccp_up
 005A  83 FB 81                                   cmp     bx,81H                  ;check for cursor left
 005D  74 24                                      je      short ccp_left
 005F  83 FB 82                                   cmp     bx,82H                  ;check for cursor down
 0062  74 27                                      je      short ccp_down
 0064  83 FB 83                                   cmp     bx,83H                  ;check for cursor right
 0067  74 2A                                      je      short ccp_right
 0069  80 E3 7F                                   and     bl,07FH
 006C  83 FB 68                                   cmp     bx,68H                  ;check for INS or button 1
 006F  74 3E                                      je      short ccp_but1
 0071  83 FB 69                                   cmp     bx,69H                  ;check for DEL or button 2
 0074  74 3E                                      je      short ccp_but2
 0076  B4 06                                      mov     ah,RS_DONE              ;recieved an unsupported key
 0078  EB 7B 90                                   jmp     exit_isr

 007B                             ccp_up:
 007B  BB 0000                                    mov     bx,0                    ;no movement on the X-axis
 007E  B9 FFF8                                    mov     cx,-8                   ;industry standard upward move
 0081  EB 18                                      jmp     short rel_move

 0083                             ccp_left:
 0083  BB FFF8                                    mov     bx,-8                   ;negative move on the X-axis
 0086  B9 0000                                    mov     cx,0                    ;no movement on the Y-axis
 0089  EB 10                                      jmp     short rel_move

 008B                             ccp_down:
 008B  BB 0000                                    mov     bx,0                    ;no movement on the X-axis
 008E  B9 0008                                    mov     cx,8                    ;industry standard down move
 0091  EB 08                                      jmp     short rel_move

 0093                             ccp_right:
 0093  BB 0008                                    mov     bx,8                    ;move right on the X-axis
 0096  B9 0000                                    mov     cx,0                    ;no movement on the Y-axis
 0099  EB 00                                      jmp     short rel_move

 009B                             rel_move:
 009B  89 1E 0028                                 mov     ds:D_REL_X,bx           ;save new relative move (X)
 009F  89 0E 002A                                 mov     ds:D_REL_Y,cx           ;save new relative move (Y)
 00A3  01 1E 0024                                 add     ds:D_ABS_X,bx           ;save new absolute position (X)
 00A7  01 0E 0026                                 add     ds:D_ABS_Y,cx           ;save new absolute position (Y)
 00AB  B6 41                                      mov     dh,T_REL16
 00AD  EB 3C                                      jmp     short give_to_parent

 00AF                             ccp_but1:
 00AF  BB 0000                                    mov     bx,0                    ;button one got pushed
 00B2  EB 05                                      jmp     short but_process

 00B4                             ccp_but2:
 00B4  BB 0001                                    mov     bx,1                    ;button two got pushed
 00B7  EB 00                                      jmp     short but_process

 00B9                             but_process:
 00B9  B8 0001                                    mov     ax,0001H                ;get the proper bit set in D_STATE
 00BC  8A CB                                      mov     cl,bl
 00BE  D2 E0                                      shl     al,cl
 00C0  A2 001C                                    mov     byte ptr ds:D_TRANSITION,al     ;record in the describe record
                                                                                          ;which button changed
```

CCP__TO__GID__FILTER

```
00C3  2E  8B 0E 0020 R                     mov     cx,word ptr cs:sav_bx     ;get the scan code and check for
00C8  F6 C1 80                              test    cl,UP_DOWN_BIT            ;push or release
00CB  74 06                                 jz      but_push

00CD                       but_release:
00CD  08 06 001D                            or      ds:D_STATE,al             ;show the release in D_STATE by
00D1  EB 08                                 jmp     short button_done         ;setting the bit

00D3                       but_push:
00D3  F6 D0                                 not     al                        ;show the push in D_STATE by
00D5  20 06 001D                            and     ds:D_STATE,al             ;clearing the bit
00D9  EB 00                                 jmp     short button_done

00DB                       button_done:
00DB  2E  A1 0020 R                         mov     ax,word ptr cs:sav_bx     ;was button pushed or released?
00DF  24 80                                 and     al,080H
00E1  0A D8                                 or      bl,al                     ;record in bx
00E3  32 FF                                 xor     bh,bh
00E5  33 C9                                 xor     cx,cx
00E7  B6 09                                 mov     dh,T_KC_BUTTON
00E9  EB 00                                 jmp     short give_to_parent

00EB                       give_to_parent:
00EB  B4 00                                 mov     ah,F_ISR                  ;Execute ISR of parent
00ED  B2 1B                                 mov     dl,V_CCPGID/6             ;source vector is this driver
00EF  8B 2E 000A                            mov     bp,ds:DH_V_PARENT         ;Get my parent's vector from my header
                                            SYSCALL
00F3  CD 6F              +                          int     HP_ENTRY
00F5                       exit_isr:
00F5  2E  8B 1E 0020 R                      mov     bx,word ptr cs:sav_bx     ;restore to keyboard ISR state
00FA  2E  8B 0E 0022 R                      mov     cx,word ptr cs:sav_cx
00FF  2E  8B 16 0024 R                      mov     dx,word ptr cs:sav_dx
0104  2E  8E 06 0026 R                      mov     es,word ptr cs:sav_es
0109  58                                    pop     ax
010A  B4 06                                 mov     ah,RS_DONE                ;Record on return
010C  CF                                    iret

                                            subttl  CCP2GID_system function

                           page
                           ;***DRIVER HEADER**************************************************
                           ;
                           ;  NAME           CCP2GID_system function
                           ;
                           ;  DESCRIPTION:  Decodes the appropriate system function.
                           ;        Supported Subfunctions are     SF_INIT
                           ;                                        SF_START
                           ;                                        SF_REPORT_STATE
                           ;                                        SF_VERSION_DESC
                           ;
                           ;  PARAMETERS
                           ;
                           ;    ON ENTRY
                           ;
                           ;    ON EXIT
                           ;
                           ;  REGISTERS ALTERED       ax, bx, di, bp
                           ;
                           ;******************************************************************
010D                       CCP2GID_SYSTEM  label   near

010D  3C 06 90 90                           cmp     al,MAX_CCP2GID_SYS_FN          ;Is the system subfunction
0111  77 0D                                 ja      short   CCP2GID_bad_sys_fn     ;within the valid range?

0113  87 EB                                 xchg    bp,bx                          ;Load the jump table index
0115  8A D8                                 mov     bl,al                          ;into bp
0117  32 FF                                 xor     bh,bh
0119  87 EB                                 xchg    bp,bx

011B  2E  FF A6 0123 R                      jmp     cs:word ptr     CCP2GID_sys_case[bp]

0120                       CCP2GID_bad_sys_fn:
0120  B4 02                                 mov     ah,RS_UNSUPPORTED              ;Return status as unsupported
0122  CF                                    iret

                           ; CCP2GID_system subfunction jumptable

0123                       CCP2GID_sys_case:
0123  012B R                                dw      word ptr        CCP2GID_sys_init        ;SF_INIT
0125  0147 R                                dw      word ptr        CCP2GID_sys_start       ;SF_START
0127  0120 R                                dw      word ptr        CCP2GID_bad_sys_fn      ;SF_REPORT_STATE
0129  0120 R                                dw      word ptr        CCP2GID_bad_sys_fn      ;SF_VERSION_DESC
= 0006                     MAX_CCP2GID_SYS_FN       equ     byte ptr     ($ - CCP2GID_sys_case - 2)

                                            subttl  CCP2GID_system function - init subfunction

                           page
                           ;***DRIVER HEADER**************************************************
                           ;
                           ;  NAME:          CCP2GID_system function - init subfunction
                           ;
                           ;  DESCRIPTION:  Initializes Describe Record and Exits, allocating a
                           ;                DS
```

```
                                ;
                                ;  PARAMETERS
                                ;
                                ;     ON ENTRY:   ah = F_SYSTEM
                                ;                 al = SF_INIT
                                ;                 bp = V_CCPGID
                                ;                 bx = last used data segment
                                ;
                                ;     ON EXIT:    ah = RS_SUCCESSFUL
                                ;                 bx = last used data segment - this drivers data segment
                                ;
                                ; REGISTERS ALTERED:       ah, bp, and ds
                                ;
                                ;*****************************************************************
 012B                          CCP2GID_sys_init label  near
 012B  06                                     push    es
 012C  56                                     push    si
 012D  57                                     push    di
 012E  51                                     push    cx
 012F  83 EB 03                               sub     bx,(CCP2GID_DESC_SIZE+15)/16
 0132  8E C3                                  mov     es,bx
 0134  BE 0177 R                              mov     si,offset cs:CCP2GID_desc_headr
 0137  FC                                     cld
 0138  33 FF                                  xor     di,di
 013A  B9 0030                                mov     cx,CCP2GID_DESC_SIZE
 013D  F3/ 2E: A4                             rep     movs byte ptr es:[di],cs:[si]
 0140  59                                     pop     cx
 0141  5F                                     pop     di
 0142  5E                                     pop     si
 0143  07                                     pop     es
 0144  B4 00                                  mov     ah,RS_SUCCESSFUL
 0146  CF                                     iret

                                        page
                                ;***DRIVER HEADER**************************************************
                                ;
                                ;  NAME:          CCP2GID_system function - start subfunction
                                ;
                                ;  DESCRIPTION:   Relinks the V_CCP driver to this driver, V_CCPGID,
                                ;                 so this driver is activated to translate cursor control
                                ;                 pad reports to mouse type movements.
                                ;
                                ;
                                ;  PARAMETERS
                                ;
                                ;     ON ENTRY    AH = F_SYSTEM
                                ;                 AL = SF_START
                                ;                 BP = V_CCPGID
                                ;
                                ;
                                ;     ON EXIT:    AH = RS_SUCCESSFUL
                                ;
                                ; REGISTERS ALTERED       ah, ds, bp
                                ;
                                ;*****************************************************************


 0147                          CCP2GID_sys_start label near
 0147  50                             push    ax
 0148  1E                             push    ds                      ;Save the ds register
 0149  B8 0000                        mov     ax,0                    ;Set the ds to the Int vector
 014C  8E D8                          mov     ds,ax
 014E  A1 01BE                        mov     ax,ds:[(4 * 6FH) + 2]   ;Get the HP_VECTOR_TABLE segment
 0151  2E: A3 0028 R                  mov     word ptr cs:top_hp_entry,ax
 0155  8E D8                          mov     ds,ax
 0157  A1 0052                        mov     ax,ds:[V_CCP+4]                 ;Get the top of the header for the
 015A  8E D8                          mov     ds,ax                           ;V_CCP driver
 015C  C7 06 000A 00A2                mov     ds:[DH_V_PARENT],V_CCPGID       ;Have V_CCP point to me
 0162  2E: A1 0028 R                  mov     ax,word ptr cs:top_hp_entry     ;load ds with HP_VECTOR_TABLE segment
 0166  8E D8                          mov     ds,ax
 0168  A1 00D0                        mov     ax,ds:[V_LHPMOUSE+4]    ;Point to the top of RAM for the mouse driver
 016B  8E D8                          mov     ds,ax
 016D  C6 06 004C 02                  mov     byte ptr ds:MSE_NUM_BUTTON,2    ;Define the number of buttons to 2.
 0172  1F                             pop     ds                              ;Restore the ds
 0173  58                             pop     ax
 0174  B4 00                          mov     ah,RS_SUCCESSFUL
 0176  CF                             iret

                                 subttl DOS-Install Code ( Returned to DOS )

                                page
 0177                          RETURN_THE_FOLLOWING_RAM_TO_DOS label    far
                               ; temporary EX-BIOS Header configuration template

 0177  E608                    CCP2GID_desc_headr      HP_SHEADER      <CCP2GID_HP_ATTR,V_CCPGID/6,V_CCPGID,CL_CCP,CL_LGID
                            ,V_LHPMOUSE,V_DOLITTLE>

 0179  001B
 017B  00A2
 017D  2000
 017F  0020
 0181  00CC
 0183  0006
 0185  00
 0186  00
```

CCP__TO__GID__FILTER

```
0187                      CCP2GID_desc
0187  0F                                  db      0FH               ;D_SOURCE
0188  00                                  db      0                 ;D_HPHIL_IO
0189  00                                  db      0                 ;D_DESC_MASK
018A  00                                  db      0                 ;D_IO_MASK
018B  00                                  db      0                 ;D_XDESC_MASK
018C  02                                  db      2                 ;D_MASK_AXIS
018D  00                                  db      0                 ;D_CLASS
018E  20                                  db      020h              ;D_PROMPTS
018F  00                                  db      0                 ;D_RESERVED
0190  00                                  db      0                 ;D_BURST_LEN
0191  00                                  db      0                 ;D_WR_REG
0192  00                                  db      0                 ;D_RD_REG
0193  01                                  db      1                 ;D_TRANSITION
0194  FF                                  db      INIT_BUT_STATE    ;D_STATE
0195  00C8                                dw      200               ;D_RESOLUTION
0197  0000                                dw      0                 ;D_SIZE_X
0199  0000                                dw      0                 ;D_SIZE_Y
019B  0000                                dw      0                 ;D_ABS_X
019D  0000                                dw      0                 ;D_ABS_Y
019F  0000                                dw      0                 ;D_REL_X
01A1  0000                                dw      0                 ;D_REL_Y
01A3  0000                                dw      0                 ;D_ACCUM_X
01A5  0000                                dw      0                 ;D_ACCUM_Y

                          ;**************************************************************;
                          ; This completes the MS-DOS device driver section
                          ; and begins the HP device driver code  (section 2)
                          ;**************************************************************;
01A7  ????                rh_off          dw      ?         ;request header pointer offset
01A9  ????                rh_seg          dw      ?         ;request header pointer segment
                          ;**************************************************************;
                          ; device strategy (required by MS-DOS 3.1)
                          ;       save es:bx (address of request header) and return
                          ;**************************************************************;
01AB                      dev_strategy    PROC    FAR
                          ;**************************************************************;
01AB  2E: 89 1E 01A7 R            mov     cs:rh_off,bx    ;save offset  of request header ptr
01B0  2E: 8C 06 01A9 R            mov     cs:rh_seg,es    ;save segment of request header ptr
01B5  CB                          ret
01B6                      dev_strategy    ENDP    FAR
                          ;**************************************************************;
                          ; device interrupt (required by MS-DOS 3.1)
                          ;       use the command from the request header block as an index
                          ;       into the table of command processing routines
                          ;**************************************************************;
01B6                      command_table   label word
                          ;**************************************************************;
01B6  0288 R                      dw      init            ;initialization
01B8  0213 R                      dw      media_check     ;media check (block only)
01BA  0213 R                      dw      build_bpb       ;build bpb   (block only)
01BC  0213 R                      dw      ioctl_in        ;ioctl input
01BE  0213 R                      dw      input           ;input (read)
01C0  0213 R                      dw      nd_input        ;non-destruct  read  (char only)
01C2  0213 R                      dw      in_stat         ;input status        (char only)
01C4  0213 R                      dw      in_flush        ;input buffer flush  (char only)
01C6  0213 R                      dw      output          ;output (write)
01C8  0213 R                      dw      out_verify      ;output (write) with verify
01CA  0213 R                      dw      out_stat        ;output status       (char only)
01CC  0213 R                      dw      out_flush       ;output buffer flush (char only)
01CE  0213 R                      dw      ioctl_out       ;ioctl output
01D0  0213 R                      dw      dev_open        ;device open
01D2  0213 R                      dw      dev_close       ;device close
01D4  0213 R                      dw      rem_media       ;removable media

                          ;**************************************************************;
01D6                      dev_int         PROC    FAR
                          ;**************************************************************;
01D6  9C                          pushf
01D7  FC                          cld                             ;preserve machine state
01D8  60                          pusha
01D9  1E                          push    ds
01DA  06                          push    es
                          ; DS is CS
01DB  0E                          push    cs
01DC  1F                          pop     ds

01DD  C4 36 01A7 R                les     si,dword ptr ds:[rh_off] ;loads es:si
01E1  26: 8A 5C 02                mov     bl,es:[si].MSD_CMD       ;get function byte
01E5  80 FB 00                    cmp     bl,MSD_INIT              ;quit if lower than
01E8  72 11                       jb      bad_cmd                  ; lowest command number
01EA  80 FB 0F                    cmp     bl,MSD_REM_MEDIA         ;quit if higher than
01ED  77 0C                       ja      bad_cmd                  ; highest command number
                          ;
                          ; command is valid; go do it
                          ;
01EF  32 FF                       xor     bh,bh
01F1  D1 E3                       shl     bx,1            ;make BX offset into table
01F3  2E: FF 97 01B6 R            call    word ptr cs:command_table[bx]
01F6  EB 10 90                    jmp     int_exit
                          ;
```

```
                                        ; unknown command routine
                                        ;
      01FB                              bad_cmd:
      01FB  C4 36 01A7 R                        les     si,dword ptr ds:[rh_off] ;reload es:si w/ header addr
      01FF  B0 03                               mov     al,MSD_UNKNOWN_CMD
      0201  B4 81                               mov     ah,MSD_ERR_STATUS        ;status word now in AX
      0203  26: 89 44 03                        mov     es:[si].MSD_STATUS,ax    ;place in request header
      0207  EB 01 90                            jmp     int_exit

                                        ; all finished

      020A                              int_exit:
      020A  C4 1E 01A7 R                        les     bx,dword ptr ds:[rh_off] ;reload es:bx w/ header addr
      020E  07                                  pop     es
      020F  1F                                  pop     ds               ;restore all preserved registers
      0210  61                                  popa
      0211  9D                                  popf
      0212  CB                                  ret
      0213                              dev_int         ENDP    FAR

                                        ;*****************************************************************;
                                        ; All MS-DOS functions except init are unsupported and do nothing  ;
                                        ;*****************************************************************;
      0213                              unsupported     PROC    NEAR
                                        ;*****************************************************************;

      0213                              media_check:
      0213                              build_bpb:
      0213                              ioctl_in:
      0213                              input:
      0213                              nd_input:
      0213                              in_stat:
      0213                              in_flush:
      0213                              output:
      0213                              out_verify:
      0213                              out_stat:
      0213                              out_flush:
      0213                              ioctl_out:
      0213                              dev_open:
      0213                              dev_close:
      0213                              ram_media:
      0213                              all_ok:
      0213  32 C0                               xor     al,al                    ;0 indicates OK
      0215  B4 01                               mov     ah,MSD_DONE_STATUS
      0217  C4 36 01A7 R                        les     si,dword ptr ds:[rh_off] ;reload es:si w/ header addr
      021B  26: 89 44 03                        mov     es:[si].MSD_STATUS,ax    ;return ok status
      021F  C3                                  ret
      0220                              unsupported     ENDP    NEAR

                                         page
                                        ;*****************************************************************;
                                        ; init  -  setup variables & establish link in HP_VECTOR_TABLE     ;
                                        ;*****************************************************************;
      0220  48 50 20 43 43 50           init_msg        db      "HP CCP2GID installed driver 2.2",0dH,0aH,"$"
            32 47 49 44 20 69
            6E 73 74 61 6C 6C
            65 64 20 64 72 69
            76 65 72 20 32 2E
            32 0D 0A 24
      0242  48 50 20 43 43 50           init_msg2       db      "HP CCP2GID installation failed",0dH,0aH,"$"
            32 47 49 44 20 69
            6E 73 74 61 6C 6C
            61 74 69 6F 6E 20
            66 61 69 6C 65 64
            0D 0A 24
      0263  48 50 20 43 43 50           init_msg3       db      "HP CCP2GID installation suceeded",0dH,0aH,"$"
            32 47 49 44 20 69
            6E 73 74 61 6C 6C
            61 74 69 6F 6E 20
            73 75 63 65 65 64
            65 64 0D 0A 24

                                        ;*****************************************************************;
      0286                              init    PROC    NEAR
                                        ;*****************************************************************;
      0286  FA                                  cli

                                        ; Put next available memory location in System Request Header
                                        ;
      0287  C4 36 01A7 R                        les     si,dword ptr ds:[rh_off]                ;es:si = header addr
      028B  8D 06 0177 R                        lea     ax,cs:RETURN_THE_FOLLOWING_RAM_TO_DOS  ;put next free loc
      028F  26: 89 44 0E                        mov     es:word ptr [si].MSD_END_OFFSET,ax     ;address in header
      0293  8C C8                               mov     ax,cs
      0295  26: 89 44 10                        mov     es:word ptr [si].MSD_END_SEG,ax

                                        ; Put the driver into the HP_VECTOR_TABLE
                                        ;
      0299  0E                                  push    cs
      029A  07                                  pop     es

                                        ; install (init ) V_CCPGID

      029B  B4 0E                               mov     ah, F_INS_FIXGETDS      ; Puts the driver in HP_VECTOR_TABLE
      029D  BB 00A2                             mov     bx, V_CCPGID            ; and calls to do F_SYSTEM+SF_INIT
      02A0  8D 3E 002A R                        lea     di, CCP2GID_DRIVER
```

CCP__TO__GID__FILTER

```
      02A4  1E                              push    ds
                                            syscall V_SYSTEM
      02A5  BD 0012           +                             mov     bp,V_SYSTEM
      02A8  CD 6F             +                             int     HP_ENTRY
      02AA  1F                              pop     ds
                                    ; start V_CCPGID
      02AB  B4 02                           mov     ah, F_SYSTEM
      02AD  B0 02                           mov     al, SF_START
      02AF  1E                              push    ds
                                            syscall V_CCPGID
      02B0  BD 00A2           +                             mov     bp,V_CCPGID
      02B3  CD 6F             +                             int     HP_ENTRY
      02B5  1F                              pop     ds
                                    ; install HP Mouse Driver whether there is an HP Mouse or not
      02B6  B4 04                           mov     ah,F_IO_CONTROL
      02B8  B0 02                           mov     al,SF_MOUSE_OVERRIDE
                                            syscall V_LHPMOUSE
      02BA  BD 00CC           +                             mov     bp,V_LHPMOUSE
      02BD  CD 6F             +                             int     HP_ENTRY
      02BF  1E                              push    ds
      02C0  0E                              push    cs
      02C1  1F                              pop     ds
                                    ; write a message on display saying driver installed
      02C2  8D 16 0220 R                    lea     dx, init_msg
      02C6  B4 09                           mov     ah, 9
      02C8  CD 21                           int     21H
      02CA  1F                              pop     ds
      02CB  FB                              sti
      02CC  E9 0213 R                       jmp     all_ok                  ;all linked so all finished
      02CF  C3                              ret
      02D0                          init    ENDP    NEAR

      02D0                          CODE    ends
                                            end


Macros:

                N a m e                 Length

MSD_HEADER                              0006
SYSCALL                                 0002

Structures and records:

                N a m e                 Width   # fields
                                        Shift   Width   Mask    Initial

DESCRIBE                                0030    0018
  D_SOURCE                              0010
  D_HPHIL_ID                            0011
  D_DESC_MASK                           0012
  D_IO_MASK                             0013
  D_XDESC_MASK                          0014
  D_MAX_AXIS                            0015
  D_CLASS                               0016
  D_PROMPTS                             0017
  D_RESERVED                            0018
  D_BURST_LEN                           0019
  D_WR_REG                              001A
  D_RD_REG                              001B
  D_TRANSITION                          001C
  D_STATE                               001D
  D_RESOLUTION                          001E
  D_SIZE_X                              0020
  D_SIZE_Y                              0022
  D_ABS_X                               0024
  D_ABS_Y                               0026
  D_REL_X                               0028
  D_REL_Y                               002A
  D_ACCUM_X                             002C
  D_ACCUM_Y                             002E
HP_SHEADER                              0010    0009
  DH_ATR                                0000
  DH_NAME_INDEX                         0002
  DH_V_DEFAULT                          0004
  DH_P_CLASS                            0006
  DH_C_CLASS                            0008
  DH_V_PARENT                           000A
  DH_V_CHILD                            000C
  DH_MAJOR                              000E
  DH_MINOR                              000F
MSD_INIT_CMD                            0017    0007
  MSD_UNIT_COUNT                        000D
  MSD_END_OFFSET                        000E
  MSD_END_SEG                           0010
  MSD_BPB_OFFSET                        0012
  MSD_BPB_SEG                           0014
  MSD_1ST_UNIT                          0016
MSD_REQ_HEADER                          0016    000A
  MSD_CMDLEN                            0000
  MSD_UNIT                              0001
```

CCP_TO_GID_FILTER

```
  MSD_CMD  . . . . . . . . . . . . . . .  0002
  MSD_STATUS . . . . . . . . . . . . . .  0003
  MSD_MEDIA  . . . . . . . . . . . . . .  000D
  MSD_TRANS  . . . . . . . . . . . . . .  000E
  MSD_COUNT  . . . . . . . . . . . . . .  0012
  MSD_START  . . . . . . . . . . . . . .  0014

Segments and Groups

                N a m e                   Size    Align    Combine Class

CGROUP . . . . . . . . . . . . . . . . .  GROUP
  CODE . . . . . . . . . . . . . . . . .  02D0    PARA     PUBLIC  'CODE'

Symbols

                N a m e                   Type    Value    Attr

ALL_OK . . . . . . . . . . . . . . . . .  L NEAR  0213     CODE
ATR_CSHARE . . . . . . . . . . . . . . .  Number  0008
ATR_DEVCFG . . . . . . . . . . . . . . .  Number  4000
ATR_HP . . . . . . . . . . . . . . . . .  Number  8000
ATR_ISR  . . . . . . . . . . . . . . . .  Number  2000
ATR_LOG  . . . . . . . . . . . . . . . .  Number  0800
BAD_CMD  . . . . . . . . . . . . . . . .  L NEAR  01FB     CODE
BUILD_BPB  . . . . . . . . . . . . . . .  L NEAR  0213     CODE
BUTTON_DONE  . . . . . . . . . . . . . .  L NEAR  00DB     CODE
BUT_PROCESS  . . . . . . . . . . . . . .  L NEAR  00B9     CODE
BUT_PUSH . . . . . . . . . . . . . . . .  L NEAR  00D3     CODE
BUT_RELEASE  . . . . . . . . . . . . . .  L NEAR  00CD     CODE
CCP2GID_BAD_SYS_FN . . . . . . . . . . .  L NEAR  0120     CODE
CCP2GID_OESC . . . . . . . . . . . . . .  L NEAR  0187     CODE
CCP2QID_DESC_HEADR . . . . . . . . . . .  L 0010  0177     CODE
CCP2GID_DESC_SIZE  . . . . . . . . . . .  Number  0030
CCP2GID_DRIVER . . . . . . . . . . . . .  F PROC  002A     CODE    Length =0010
CCP2GID_HP_ATTR  . . . . . . . . . . . .  Number  E808
CCP2GID_INSTALLED  . . . . . . . . . . .  L FAR   0000     CODE
CCP2GID_ISR  . . . . . . . . . . . . . .  L NEAR  003A     CODE
CCP2GID_SYSTEM . . . . . . . . . . . . .  L NEAR  010D     CODE
CCP2GID_SYS_CASE . . . . . . . . . . . .  L NEAR  0123     CODE
CCP2QID_SYS_INIT . . . . . . . . . . . .  L NEAR  012B     CODE
CCP2GID_SYS_START  . . . . . . . . . . .  L NEAR  0147     CODE
CCP2GID_UNSUPPORTED  . . . . . . . . . .  L NEAR  0037     CODE
CCP_BUTI . . . . . . . . . . . . . . . .  L NEAR  00AF     CODE
CCP_BUT2 . . . . . . . . . . . . . . . .  L NEAR  00B4     CODE
CCP_DOWN . . . . . . . . . . . . . . . .  L NEAR  008B     CODE
CCP_LEFT . . . . . . . . . . . . . . . .  L NEAR  0083     CODE
CCP_RIGHT  . . . . . . . . . . . . . . .  L NEAR  0093     CODE
CCP_UP . . . . . . . . . . . . . . . . .  L NEAR  007B     CODE
CL_CCP . . . . . . . . . . . . . . . . .  Number  2000
CL_LGID  . . . . . . . . . . . . . . . .  Number  0020
COMMAND_TABLE  . . . . . . . . . . . . .  L WORD  01B6     CODE
DESCRIBE_SIZE  . . . . . . . . . . . . .  Number  0030
DEV_CLOSE  . . . . . . . . . . . . . . .  L NEAR  0213     CODE
DEV_INT  . . . . . . . . . . . . . . . .  F PROC  01D8     CODE    Length =003D
DEV_OPEN . . . . . . . . . . . . . . . .  L NEAR  0213     CODE
DEV_STRATEGY . . . . . . . . . . . . . .  F PROC  01AB     CODE    Length =000B
D_ADDR_MASK  . . . . . . . . . . . . . .  Number  000F
D_BUFFER . . . . . . . . . . . . . . . .  Alias   D_SIZE_X
D_CCP_STATE  . . . . . . . . . . . . . .  Number  001E
D_CLASS_CURRENT  . . . . . . . . . . . .  Number  00F0
D_CLASS_DEFAULT  . . . . . . . . . . . .  Number  000F
D_REMAINDER_ACCUM  . . . . . . . . . . .  Alias   D_ACCUM_X
D_SAMPLE_ABSOLUTE  . . . . . . . . . . .  Alias   D_ABS_X
D_SAMPLE_RELATIVE  . . . . . . . . . . .  Alias   D_REL_X
D_SIZE . . . . . . . . . . . . . . . . .  Alias   D_SIZE_X
D_TYPE_MASK  . . . . . . . . . . . . . .  Number  00F0
EXIT_ISR . . . . . . . . . . . . . . . .  L NEAR  00F5     CODE
F_INS_FIXGETDS . . . . . . . . . . . . .  Number  000E
F_IO_CONTROL . . . . . . . . . . . . . .  Number  0004
F_ISR  . . . . . . . . . . . . . . . . .  Number  0000
F_SYSTEM . . . . . . . . . . . . . . . .  Number  0002
GIVE_TO_PARENT . . . . . . . . . . . . .  L NEAR  00EB     CODE
HP_ENTRY . . . . . . . . . . . . . . . .  Number  008F
INIT . . . . . . . . . . . . . . . . . .  N PROC  02B8     CODE    Length =004A
INIT_BUT_STATE . . . . . . . . . . . . .  Number  00FF
INIT_MSG . . . . . . . . . . . . . . . .  L BYTE  0220     CODE
INIT_MSG2  . . . . . . . . . . . . . . .  L BYTE  0242     CODE
INIT_MSG3  . . . . . . . . . . . . . . .  L BYTE  0283     CODE
INPUT  . . . . . . . . . . . . . . . . .  L NEAR  0213     CODE
INT_EXIT . . . . . . . . . . . . . . . .  L NEAR  020A     CODE
IN_FLUSH . . . . . . . . . . . . . . . .  L NEAR  0213     CODE
IN_STAT  . . . . . . . . . . . . . . . .  L NEAR  0213     CODE
IOCTL_IN . . . . . . . . . . . . . . . .  L NEAR  0213     CODE
IOCTL_OUT  . . . . . . . . . . . . . . .  L NEAR  0213     CODE
MAX_CCP2GID_SYS_FN . . . . . . . . . . .  E BYTE  0008
MEDIA_CHECK  . . . . . . . . . . . . . .  L NEAR  0213     CODE
MSD_DONE_STATUS  . . . . . . . . . . . .  Number  0001
MSD_ERR_STATUS . . . . . . . . . . . . .  Number  0081
MSD_INIT . . . . . . . . . . . . . . . .  Number  0000
MSD_REM_MEDIA  . . . . . . . . . . . . .  Number  000F
MSD_UNKNOWN_CMD  . . . . . . . . . . . .  Number  0003
MSE_NUM_BUTTON . . . . . . . . . . . . .  Number  004C
ND_INPUT . . . . . . . . . . . . . . . .  L NEAR  0213     CODE
```

CCP__TO__GID__FILTER

```
OUTPUT . . . . . . . . . . . . . . . . . . . L NEAR  0213   CODE
OUT_FLUSH  . . . . . . . . . . . . . . . . . L NEAR  0213   CODE
OUT_STAT . . . . . . . . . . . . . . . . . . L NEAR  0213   CODE
OUT_VERIFY . . . . . . . . . . . . . . . . . L NEAR  0213   CODE
REL_MOVE . . . . . . . . . . . . . . . . . . L NEAR  009B   CODE
REM_MEDIA  . . . . . . . . . . . . . . . . . L NEAR  0213   CODE
RETURN_THE_FOLLOWING_RAM_TO_DOS. . . . . . . L FAR   0177   CODE
RH_OFF . . . . . . . . . . . . . . . . . . . L WORD  01A7   CODE
RH_SEG . . . . . . . . . . . . . . . . . . . L WORD  01A9   CODE
RS_DONE  . . . . . . . . . . . . . . . . . . Number  0008
RS_SUCCESSFUL. . . . . . . . . . . . . . . . Number  0000
RS_UNSUPPORTED . . . . . . . . . . . . . . . Number  0002
SAV_BX . . . . . . . . . . . . . . . . . . . L NEAR  0020   CODE
SAV_CX . . . . . . . . . . . . . . . . . . . L NEAR  0022   CODE
SAV_DX . . . . . . . . . . . . . . . . . . . L NEAR  0024   CODE
SAV_ES . . . . . . . . . . . . . . . . . . . L NEAR  0026   CODE
SF_MOUSE_OVERRIDE. . . . . . . . . . . . . . Number  0002
SF_START . . . . . . . . . . . . . . . . . . Number  0002
TOP_HP_ENTRY . . . . . . . . . . . . . . . . L NEAR  0028   CODE
T_KC_BUTTON. . . . . . . . . . . . . . . . . Number  0009
T_REL16  . . . . . . . . . . . . . . . . . . Number  0041
UNSUPPORTED. . . . . . . . . . . . . . . . . N PROC  0213   CODE    Length =000D
UP_DOWN_BIT  . . . . . . . . . . . . . . . . Number  0080
V_CCP  . . . . . . . . . . . . . . . . . . . Number  004E
V_CCPGID . . . . . . . . . . . . . . . . . . Number  00A2
V_DOLITTLE . . . . . . . . . . . . . . . . . Number  0006
V_LHPMOUSE . . . . . . . . . . . . . . . . . Number  00CC
V_SYSTEM . . . . . . . . . . . . . . . . . . Number  0012

 43048 Bytes free

 Warning Severe
 Errors  Errors
 0       0


ALL_OK . . . . . . . . . . . . . .  691#   772
ATR_CSHARE . . . . . . . . . . . .   69#   200
ATR_DEVCFG . . . . . . . . . . . .   70#   200
ATR_HP . . . . . . . . . . . . . .   71#   200
ATR_ISR. . . . . . . . . . . . . .   72#   200
ATR_LOG. . . . . . . . . . . . . .   73#   200

BAD_CMD. . . . . . . . . . . . . .  640    642    653#
BUILD_BPB. . . . . . . . . . . . .  610    677#
BUTTON_DONE. . . . . . . . . . . .  380    385    387#
BUT_PROCESS. . . . . . . . . . . .  361    365    367#
BUT_PUSH . . . . . . . . . . . . .  376    382#
BUT_RELEASE. . . . . . . . . . . .  378#

CCP2GID_BAD_SYS_FN . . . . . . . .  435    444#   453    454
CCP2GID_DESC . . . . . . . . . . .  560#
CCP2GID_DESC_HEADR . . . . . . . .  488    550#
CCP2GID_DESC_SIZE. . . . . . . . .  199#   486    491
CCP2GID_DRIVER . . . . . . . . . .  247#   257    743
CCP2GID_HP_ATTR. . . . . . . . . .  200#   550
CCP2GID_INSTALLED. . . . . . . . .  209#
CCP2GID_ISR. . . . . . . . . . . .  249    303#
CCP2GID_SYSTEM . . . . . . . . . .  252    432#
CCP2GID_SYS_CASE . . . . . . . . .  442    450#   455
CCP2GID_SYS_INIT . . . . . . . . .  451    481#
CCP2GID_SYS_START. . . . . . . . .  452    524#
CCP2GID_UNSUPPORTED. . . . . . . .  251    253#
CCP_BUT1 . . . . . . . . . . . . .  325    359#
CCP_BUT2 . . . . . . . . . . . . .  327    363#
CCP_DOWN . . . . . . . . . . . . .  320    341#
CCP_LEFT . . . . . . . . . . . . .  318    336#
CCP_RIGHT. . . . . . . . . . . . .  322    346#
CCP_UP . . . . . . . . . . . . . .  316    331#
CGROUP . . . . . . . . . . . . . .  205
CL_CCP . . . . . . . . . . . . . .   74#   550
CL_LGID. . . . . . . . . . . . . .   75#   550
CODE . . . . . . . . . . . . . . .  205    206#   206    207    776
COMMAND_TABLE. . . . . . . . . . .  606#   648

DESCRIBE . . . . . . . . . . . . .   77#   112    114
DESCRIBE_SIZE. . . . . . . . . . .  114#
DEV_CLOSE. . . . . . . . . . . . .  622    689#
DEV_INT. . . . . . . . . . . . . .  228    626#   669
DEV_OPEN . . . . . . . . . . . . .  621    688#
DEV_STRATEGY . . . . . . . . . . .  227    595#   600
DH_ATR . . . . . . . . . . . . . .   49#
DH_C_CLASS . . . . . . . . . . . .   53#
DH_MAJOR . . . . . . . . . . . . .   56#
DH_MINOR . . . . . . . . . . . . .   57#
DH_NAME_INDEX. . . . . . . . . . .   50#
DH_P_CLASS . . . . . . . . . . . .   52#
DH_V_CHILD . . . . . . . . . . . .   55#
DH_V_DEFAULT . . . . . . . . . . .   51#
DH_V_PARENT. . . . . . . . . . . .   54#   399    534
D_ABS_X. . . . . . . . . . . . . .  106#   118    354
```

CCP_TO_GID_FILTER

D_ABS_Y	107#	355			
D_ACCUM_X	110#	120			
D_ACCUM_Y	111#				
D_ADDR_MASK	125#				
D_BUFFER	121#				
D_BURST_LEN	96#				
D_CCP_STATE	116#				
D_CLASS	89#				
D_CLASS_CURRENT	122#				
D_CLASS_DEFAULT	123#				
D_DESC_MASK	85#				
D_HPHIL_ID	84#				
D_IO_MASK	86#				
D_MAX_AXIS	88#				
D_PROMPTS	92#				
D_RD_REG	100#				
D_REL_X	108#	119	352		
D_REL_Y	109#	353			
D_REMAINDER_ACCUM	120#				
D_RESERVED	95#				
D_RESOLUTION	103#				
D_SAMPLE_ABSOLUTE	118#				
D_SAMPLE_RELATIVE	119#				
D_SIZE	117#				
D_SIZE_X	104#	117	121		
D_SIZE_Y	105#				
D_SOURCE	82#				
D_STATE	102#	116	379	384	
D_TRANSITION	101#	371			
D_TYPF_MASK	126#				
D_WR_REG	99#				
D_XDESC_MASK	87#				
EXIT_ISR	32#	402#			
F_INS_FIXGETDS	128#	741			
F_IO_CONTROL	129#	758			
F_ISR	131#	248	397		
F_SYSTEM	132#	250	750		
GIVE_TO_PARENT	357	394	396#		
HP_ENTRY	60#	401	747	755	762
HP_SHEADER	48#	58	78		
INIT	608	722#	774		
INIT_BUT_STATE	197#	574			
INIT_MSG	703#	767			
INIT_MSG2	709#				
INIT_MSG3	715#				
INPUT	612	679#			
INT_EXIT	649	658	662#		
IN_FLUSH	615	682#			
IN_STAT	614	681#			
IOCTL_IN	611	678#			
IOCTL_OUT	620	687#			
MAX_CCP2GID_SYS_FN	434	455#			
MEDIA_CHECK	609	676#			
MSD_1ST_UNIT	174#				
MSD_BPB_OFFSET	172#				
MSD_BPB_SEG	173#				
MSD_CMD	150#	638			
MSD_CMDLEN	148#				
MSD_COUNT	160#				
MSD_DONE_STATUS	181#	693			
MSD_END_OFFSET	170#	730			
MSD_END_SEG	171#	732			
MSD_ERR_STATUS	180#	656			
MSD_HEADER	224				
MSD_INIT	177#	639			
MSD_INIT_CMD	164#	175			
MSD_MEDIA	157#				
MSD_REM_MEDIA	179#	641			
MSD_REQ_HEADER	147#	162			
MSD_START	161#				
MSD_STATUS	151#	657	695		
MSD_TRANS	158#				
MSD_UNIT	149#				
MSD_UNIT_COUNT	169#				
MSD_UNKNOWN_CMD	178#	655			
MSE_NUM_BUTTON	198#	539			
ND_INPUT	613	680#			
OUTPUT	616	683#			
OUT_FLUSH	619	686#			
OUT_STAT	618	685#			
OUT_VERIFY	617	684#			
REL_MOVE	334	339	344	349	351#
REM_MEDIA	623	690#			

CCP__TO__GID__FILTER

RETURN_THE_FOLLOWING_RAM_TO_DOS	5470	729					
RH_OFF	5890	597	637	654	663	694	728
RH_SEG	5900	598					
RS_DONE	1830	328	408				
RS_SUCCESSFUL	1840	497	542				
RS_UNSUPPORTED	1850	254	445				
SAV_BX	2370	306	374	380	403		
SAV_CX	2380	307	404				
SAV_DX	2390	308	405				
SAV_ES	2400	309	406				
SF_MOUSE_OVERRIDE	1300	759					
SF_START	1330	751					
SYSCALL	400	745	753	760			
TOP_HP_ENTRY	2410	530	535				
T_KC_BUTTON	1870	393					
T_REL18	1880	356					
UNSUPPORTED	6740	697					
UP_DOWN_BIT	1960	375					
V_CCP	1940	532					
V_CCPGID	1910	398	534	550	550	742	754
V_DOLITTLE	1900	550					
V_LHPMOUSE	1920	537	550	761			
V_SYSTEM	1930	746					

158 Symbols

54092 Bytes Free

Application Resident EX-BIOS Driver

This example demonstrates the use of an application resident EX-BIOS driver. The driver utilizes the Touchscreen logical device driver V_LTOUCH and its associated event driver V_EVENT_TOUCH.

The driver utilizes V_LTOUCH to move the cursor around the screen. V_LTOUCH returns the current row and column address of the point at which the screen is being touched. The example driver in turn utilizes the STD-BIOS Video driver (INT 10H) to change to the position of the displayed cursor to match the screen coordinates returned by V_LTOUCH.

This driver also utilizes the button state data returned by V_LTOUCH. When the screen is touched (a button make) the driver changes the shape of the cursor from an underline to a box or full character cell. The shape of the cursor is restored to an underline when the finger is removed (a button break).

Notice in the initialization section of the code that the CS:IP of the driver's service routine (TOUCH_HANDLER) and the driver's DS are substituted into the V_EVENT_TOUCH vector in the HP_VECTOR_TABLE. The existing contents of that vector are returned by the function. The driver stores these values in its data area and restores them when the driver terminates (a " character is typed at the keyboard). All HP_VECTOR_TABLE vectors that are replaced with application program resident drivers should restore the original values in the vector when the application program terminates.

The listing for this driver can be found in Chapter 4.

Non-HP-HIL Input Devices

The next program listing is an example of how to integrate non-HP-HIL input devices into the Input System. This driver interfaces to an RS-232 mouse. It converts data frames received from the mouse into GID motion and button ISR Event Records. It integrates itself into the Input System by calling the V_SINPUT driver once these ISR Event Records have been constructed.

The PGID driver is the physical device driver for all devices inputting graphic motion and button state data. The initialization code must create a PGID driver for the V_SINPUT to pass the ISR Event Record. It builds a driver header and physical describe record, allocates a free HP_VECTOR_TABLE vector, and installs the PGID driver with V_LHPMOUSE as its parent driver.

The driver is structured as a DOS installable device driver. The COM port the mouse is connected to can be specified in the CONFIG.SYS command line.

RS-232 Mouse Driver

```
                                  .286c
                                  .LFCOND
                                  PAGE 59,132
                                  TITLE RS-232 MOUSE DRIVER
                                  SUBTTL PREFACE
                                  ;*****************************************************************
                                  ;*                                                               *
                                  ;*                  RS-232 MOUSE DRIVER EXAMPLE                  *
                                  ;*                                                               *
                                  ;*****************************************************************

                                  ;*****************************************************************
                                  ;*                          DESCRIPTION                          *
                                  ;*****************************************************************

                                  ; This driver illustrates the integration of non-HP-HIL devices into the
                                  ; HP Vectra Input System.  This driver supports any mouse with an RS-232
                                  ; interface, such as the MOUSE SYSTEMS mouse.  The driver is installed as
                                  ; an MS-DOS device driver at boot time.

                                  ;The command line DEVICE=EXAMPLE.SYS [/n] should be entered in the
                                  ;CONFIG.SYS file in the root directory of the boot drive.  If the optional
                                  ;COM port number, /n is not included in the command line, the driver will
                                  ;attempt to install the mouse on COM1.  If the optional COM port number is
                                  ;present in the command line, the driver will attempt to install the mouse
                                  ;on that COM port number.  The driver checks to make sure the port is present
                                  ;and will issue an error message if a non-existent port number is specified.

                                  ;*****************************************************************
                                  ;*                          CHANGE LOG                           *
                                  ;*****************************************************************

                                  ;Revision A.01.01 - 12/02/85   SMM

                                  ;*****************************************************************

                                  SUBTTL EQUATES, RECORDS, AND DATA STRUCTURES
                                  PAGE
                                  ;*****************************************************************
                                  ;*****************************************************************
                                  ;                  EQUATES AND DATA STRUCTURES
                                  ;*****************************************************************
                                  ;*****************************************************************

                                  ;*********************** GENERAL EQUATES ***********************

= 0000                            FALSE          EQU 0
=-0001                            TRUE           EQU NOT FALSE
=                                 DEBUG          EQU TRUE

                                  ;** MS-DOS INSTALLABLE DEVICE DRIVER EQUATES, RECORDS, AND STRUCTURES **

                                  ;STRUCTURES

                                  REQ_HEADER     STRUC                     ;Initialization Request Header
                                                                           ;structure definition
0000  ??                          RH_LENGTH      DB ?                      ;Length of Request Header
0001  ??                          RH_UNIT_CODE   DB ?                      ;Unit code
0002  ??                          RH_CMD_CODE    DB ?                      ;Command code
0003  ????                        RH_STATUS      DW ?                      ;Returned status
0005  ????????????????            RH_RESERVED    DQ ?                      ;Reserved for MS-DOS
000D  ??                          RH_UNIT_CNT    DB ?                      ;Unit count
000E  ????                        RH_END_OFF     DW ?                      ;Offset of ending address
0010  ????                        RH_END_SEG     DW ?                      ;Segment of ending address
0012  ????????                    RH_BPB         DD ?                      ;BPB Pointer (not used)
0016  ??                          RH_DRIV        DB ?                      ;Drive code (not used)

0017                              REQ_HEADER     ENDS

= 0012                            RH_CMD_LINE    EQU DWORD PTR RH_BPB      ;On INIT entry, points to CONFIG.SYS
                                                                           ;command line (i.e. all after DEVICE=)

                                  ;RECORDS

                                  ATTR RECORD  DEV:1, IOCTL:1, IBM:1, X:1, OCREM:1, Y:6, SPEC:1, CLK:1, NUL:1, STDO:1, STDI:1

                                                                           ;DEV = 1 for character device, 0 for block device
                                                                           ;IOCTL = 1 if IOCTL commands are supported
                                                                           ;IBM = 1 if block device is in non-IBM format
                                                                           ;X = Not used
                                                                           ;OCREM = 1 if character device supports open and
                                                                           ;        close commands, 1 if block device has
                                                                           ;        removable media
                                                                           ;Y = Not used
                                                                           ;SPEC = 1 if INT 29H fast console I/O is installed
                                                                           ;CLK = 1 if device is a clock device
                                                                           ;NUL = 1 if device is a nul device
                                                                           ;STDO = 1 if device is the Standard Output device
                                                                           ;STDI = 1 if device is the Standard Input device

                                  STATUS  RECORD ERROR:1, Z:5, BUSY:1, DONE:1, ERR_TYPE:8
```

```
                                                                ;ERROR = 1 if error condition detected
                                                                ;Z = Not used
                                                                ;BUSY = 1 if device busy
                                                                ;DONE = 1 when command completed
                                                                ;ERR_TYPE = Error type   See equates next

                        ;EQUATES

                        ;Error codes   Returned as part of status word defined above

    = 0000              MSD_WRITE_PROT          EQU 00H         ;write protect
    = 0001              MSD_UNKNOWN_UNIT        EQU 01H         ;unknown unit
    = 0002              MSD_NOT_RDY             EQU 02H         ;device not ready
    = 0003              MSD_UNKNOWN_CMD         EQU 03H         ;unknown command
    = 0004              MSD_CRC_ERROR           EQU 04H         ;CRC error
    = 0005              MSD_BAD_LENGTH          EQU 05H         ;bad driver request structure length
    = 0006              MSD_SEEK_ERROR          EQU 06H         ;seek error
    = 0007              MSD_UNKNOWN_MEDIA       EQU 07H         ;unknown media
    = 0008              MSD_SEC_NOT_FOUND       EQU 08H         ;sector not found
    = 0009              MSD_PAPER_OUT           EQU 09H         ;paper out
    = 000A              MSD_WRITE_FAULT         EQU 0AH         ;write fault
    = 000B              MSD_READ_FAULT          EQU 0BH         ;read fault
    = 000C              MSD_GEN_FAILURE         EQU 0CH         ;general failure

                        ;Commands

    = 0000              MSD_INIT        EQU 00H         ;Initialize
    = 0001              MSD_MEDIA_CHK   EQU 01H         ;Media check
    = 0002              MSD_BLD_BPB     EQU 02H         ;Build BIOS Parameter Block (BPB)
    = 0003              MSD_IOCTL_IN    EQU 03H         ;IOCTL input
    = 0004              MSD_INPUT       EQU 04H         ;Input from device
    = 0005              MSD_IN_NOWAIT   EQU 05H         ;Non-destructive, no-wait input
    = 0006              MSD_IN_STATUS   EQU 06H         ;Return status of input device
    = 0007              MSD_IN_FLUSH    EQU 07H         ;Flush input buffer
    = 0008              MSD_OUTPUT      EQU 08H         ;Output to device
    = 0009              MSD_OUT_VERIFY  EQU 09H         ;Output with verify to device
    = 000A              MSD_OUT_STATUS  EQU 0AH         ;Return status of output device
    = 000B              MSD_OUT_FLUSH   EQU 0BH         ;Flush output buffer
    = 000C              MSD_IOCTL_OUT   EQU 0CH         ;IOCTL output
    = 000D              MSD_DEV_OPEN    EQU 0DH         ;Open device
    = 000E              MSD_DEV_CLOSE   EQU 0EH         ;Close device
    = 000F              MSD_REM_MEDIA   EQU 0FH         ;Removable media check

                        ;MS-DOS equates

    = 0009              PRINT_STR       EQU 09H         ;MS-DOS print string function number
    = 0021              DOS_ENTRY       EQU 21H         ;MS-DOS interrupt

                        ;ASCII equates

    = 000A              LF              EQU 0AH
    = 000D              CR              EQU 0DH

                        ;***********   EX-BIOS DRIVER EQUATES, RECORDS, AND STRUCTURES   **********

                        ;STRUCTURES

                        HP_HEADER       STRUC           ;HP Driver Header

    0000  0000          DH_ATR          DW 0            ;Driver attribute
    0002  0000          DH_NAME_INDEX   DW 0            ;Index number for driver string
    0004  0000          DH_V_DEFAULT    DW 0            ;????
    0006  0000          DH_P_CLASS      DW 0            ;Driver parent class
    0008  0000          DH_C_CLASS      DW 0            ;Driver child class
    000A  0000          DH_V_PARENT     DW 0            ;Vector number of driver's parent
    000C  0000          DH_V_CHILD      DW 0            ;Vector number of driver's child
    000E  00            DH_MAJOR        DB 0            ;Major address of device
    000F  00            DH_MINOR        DB 0            ;Minor address of device

    0010                HP_HEADER       ENDS


                        DESCRIBE        STRUC           ;Physical describe record

    0000  ??            D_SOURCE        DB ?            ;Upper nibble contains GID type
                                                        ;Lower nibble HP-HIL address
    0001  ??            D_HPHIL_ID      DB ?            ;Device ID byte returned by HP-HIL device
    0002  ??            D_DESC_MASK     DB ?            ;????
    0003  ??            D_IO_MASK       DB ?            ;I/O descriptor byte from device
    0004  ??            D_XDESC_MASK    DB ?            ;Extended descriptor byte from device
    0005  ??            D_MAX_AXIS      DB ?            ;Maximum number of axes reported by device
    0006  ??            D_CLASS         DB ?            ;Device class
                                                        ;Upper nibble contains current class
                                                        ;Lower nibble contain default class
    0007  ??            D_PROMPTS       DB ?            ;Number of buttons/prompts
                                                        ;Upper nibble contains number of prompts
                                                        ;Lower nibble contains number of buttons
    0008  ??            D_RESERVED      DB ?            ;Reserved
    0009  ??            D_BURST_LEN     DB ?            ;Maximum burst length
    000A  ??            D_WR_REG        DB ?            ;Number of write registers supported
    000B  ??            D_RD_REG        DB ?            ;Number of read registers supported
    000C  ??            D_TRANSITION    DB ?            ;Transitions reported per button
    000D  ??            D_STATE         DB ?            ;Current state of buttons
    000E  ????          D_RESOLUTION    DW ?            ;Counts/cm returned by device
```

```
        0010  ????                D_SIZE_X        DW ?                      ;Maximum count along X axis in units of resolution.
        0012  ????                D_SIZE_Y        DW ?                      ;Maximum count along Y axis in units of resolution.
        0014  ????                D_ABS_X         DW ?                      ;Absolute data device X motion.
        0016  ????                D_ABS_Y         DW ?                      ;Absolute data device Y motion.
        0018  ????                D_REL_X         DW ?                      ;Relative data device X motion.
        001A  ????                D_REL_Y         DW ?                      ;Relative data device Y motion
        001C  ????                D_ACCUM_X       DW ?                      ;X axis scaling accumulator
        001E  ????                D_ACCUM_Y       DW ?                      ;Y axis scaling accumulator

        0020                      DESCRIBE        ENDS

        = 004C                    MSE_NUM_BUTTON        equ     D04CH     ;Offset of number of button in mouse RAM

                                  ;RECORDS

                                  HP_ATTR RECORD HP:1, DEVCFG:1, ISR:1, ENTRY:1, TYPE:3, STR:1, MAP_CALL:1, A:1, SUBADD:2, PS
                        HARE:1, CSHARE:1, ROM:1, B:1

                                  ;EQUATES

                                  ;EX-BIOS driver vector addresses and driver function numbers

        = 0006                    V_DOLITTLE      EQU 0006H                 ;DOLITTLE driver vector address (NUL driver)

        = 0012                    V_SYSTEM        EQU 0012H                 ;SYSTEM driver vector address
        = 0004                    F_INS_BASEHPVT  EQU 04H
        = 000A                    F_INS_XCHGFREE  EQU 0AH

        = 002A                    V_SINPUT        EQU 002AH                 ;INPUT driver vector address
        = 0000                    F_ISR           EQU 00H
        = 0002                    F_SYSTEM        EQU 02H
        = 0004                    F_IO_CONTROL    EQU 04H

        = 000C                    F_INQUIRE_ENTRY EQU 0CH                   ; inquire about PGID CS IP

        = 00CC                    V_LHPMOUSE      EQU 00CCH                 ;LHPMOUSE driver vector address
        = 0002                    SF_MOUSE_OVERRIDE EQU 02H

        = 006F                    HP_ENTRY        EQU 6FH                   ;EX-BIOS interrupt number

                                  ;ISR Event Record data types

        = 0009                    T_KC_BUTTON     EQU 09H                   ;Button data type.
        = 0040                    T_REL08         EQU 40H
        = 0041                    T_REL16         EQU 41H                   ;16 bit relative motion data type
        = 0042                    T_ABS08         EQU 42H
        = 0043                    T_ABS16         EQU 43H

                                  ;EX-BIOS Return Status Codes

        = 0000                    RS_SUCCESSFUL   EQU 00H
        = 0002                    RS_UNSUPPORTED  EQU 02H
        = 0006                    RS_DONE         EQU 06H
        = 00FE                    RS_FAIL         EQU 0FEH
        = 00F6                    RS_NO_VECTOR    EQU 0F6H

                                  ;***************************************************************

                                  SUBTTL CODE SEGMENT
                                  PAGE
                                  ;***************************************************************
                                  ;***************************************************************
                                  ;                         CODE SEGMENT
                                  ;***************************************************************
                                  ;***************************************************************

        0000                      CODE      SEGMENT PUBLIC 'CODE'

                                            ASSUME  CS:CODE, DS:NOTHING
        0000                                ORG     0                   ;Must be org'd at 0 to be a device driver

        0000                      DEV_DRIVER PROC FAR

                                  ;********************  MS-DOS DEVICE DRIVER HEADER  ****************

        0000  FF FF FF FF                         DD -1                 ;Link list entry   Must be set to -1
        0004  8000                DRIVER_ATTR     DW ATTR<1,0,0,0,0,0,0,0,0,0> ;Driver attribute
        0006  0265 R              STRAT_ENT       DW OFFSET DEV_STRATEGY  ;Device strategy entry point
        0008  0270 R              INT_ENT         DW OFFSET DEV_INTERRUPT ;Device interrupt entry point
        000A  20 32 33 32 4D 53   DRIVER_NAME     DB ' 232MSE '
              45 20

                                  ;**********  EX-BIOS DRIVER HEADER AND PHYSICAL DESCRIBE RECORD  ********

        0020                                      ORG 20H               ;Make sure its paragraph aligned

        = AC18                    DEV_ATTR        EQU HP_ATTR<1,0,1,0,6,0,0,0,1,1,0,0>
        0020  AC18                DEV_HEADER      HP_HEADER<DEV_ATTR,3,0,0,0,V_LHPMOUSE,V_DOLITTLE,0,0>
        0022  0003
        0024  0000
        0026  0000
        0028  0000
        002A  00CC
```

```
        002C  0006
        002E  00
        002F  00

        0030  02                    DEV_DESCRIBE    DESCRIBE <2,0,0,0,0,2,0,20H,0,0,0,0,1,0FFH,200D,0,0,0,0,0,0,0,0>
        0031  00
        0032  00
        0033  00
        0034  00
        0035  02
        0036  00
        0037  20
        0038  00
        0039  00
        003A  00
        003B  00
        003C  01
        003D  FF
        003E  00C8
        0040  0000
        0042  0000
        0044  0000
        0046  0000
        0048  0000
        004A  0000
        004C  0000
        004E  0000


                                    ;*****************************************************************
                                    ;*                     CODE SEGMENT RELATIVE DATA AREA           *
                                    ;*****************************************************************

                                    ;**************** DATA AREA FOR MS-DOS DRIVER PORTION ***************

        0050  0000                  REQ_HDR_OFF     DW 0                    ;Storage for offset of device strategy header.
        0052  0000                  REQ_HDR_SEG     DW 0                    ;Storage for segment of device strategy header

        0054  52 53 2D 32 33 32     SIGN_ON_MSG     DB 'RS-232 INPUT SYSTEM MOUSE DRIVER
              20 49 4E 50 55 54
              20 53 59 53 54 45
              4D 20 4D 4F 55 53
              45 20 44 52 49 56
              45 52 20 20
        0076  28 43 29 43 6F 70                     DB '(C)Copyright Hewlett-Packard 1985',CR,LF
              79 72 69 67 68 74
              20 48 65 77 6C 65
              74 74 2D 50 61 63
              6B 61 72 64 20 31
              39 38 35 0D 0A
        0099  56 65 72 73 69 6F     VERSION_LAB     DB 'Version A.01.01',CR,LF,'$'
              6E 20 41 2E 30 31
              2E 30 31 0D 0A 24
        = 0010                      VERSION_LEN     EQU $-VERSION_LAB-2
        00AB  4D 6F 75 73 65 20     OK_MSG          DB 'Mouse installed on COM'
              69 6E 73 74 61 6C
              6C 65 64 20 6F 6E
              20 43 4F 4D
        00C1  30 3A 0D 0A 0D 0A     COM_MSG         DB '0:',CR,LF,CR,LF,'$'
              24
        00C8  53 70 65 63 69 66     NO_PORT_MSG     DB 'Specified COM port not present.  Driver not installed.',CR,LF,CR,LF,
              69 65 64 20 43 4F
              4D 20 70 6F 72 74
              20 6E 6F 74 20 70
              72 65 73 65 6E 74
              2E 20 20 44 72 69
              76 65 72 20 6E 6F
              74 20 69 6E 73 74
              61 6C 6C 65 64 2E
              0D 0A 0D 0A 24
        0103  55 6E 61 62 6C 65     NO_VECTOR       DB 'Unable to install PGID driver.',CR,LF,'$'
              20 74 6F 20 69 6E
              73 74 61 6C 6C 20
              50 47 49 44 20 64
              72 69 76 65 72 2E
              0D 0A 24

        0124  0000                  STACK_PTR       DW 0                    ;Storage for existing stack frame.
        0126  0000                  STACK_SEG       DW 0

        0128  0000                  COM_NUMBER      DW 0                    ;Offset into COM port base address table
                                                                            ;found at 0040:0000H.
        012A  0030                  INT_TABLE       DW 0CH * 4              ;COM1 port interrupt.
        012C  002C                                  DW 0BH * 4              ;COM2 port interrupt.
        012E  0030                                  DW 0CH * 4              ;COM3 port interrupt - set as appropriate.
        0130  002C                                  DW 0BH * 4              ;COM4 port interrupt - set as appropriate.
        0132  FFEF                  MASK_TABLE      DW NOT 01H SHL 4        ;COM1 interrupt mask (IRQ4).
        0134  FFF7                                  DW NOT 01H SHL 3        ;COM2 interrupt mask (IRQ3).
        0136  FFEF                                  DW NOT 01H SHL 4        ;COM3 interrupt mask (IRQ4).
        0138  FFF7                                  DW NOT 01H SHL 3        ;COM4 interrupt mask (IRQ3).

        013A  0000                  FRAME_COUNT     DW 0                    ;Frame counter for mouse data packet.
        013C     05 [               TEMP_BUFFER     DB 5 DUP (0)            ;Temporary buffer for mouse data bytes.
                         00
                             ]
```

RS-232 Mouse Driver

```
0141  87                          LAST_SYNCH     DB 87H                  ;Copy of last synch byte

0142     0E [                     HPHIL_TABLE    DB 14 DUP (0)           ;HP-HIL configuration table
                00
                   ]

0150  00                          HPHIL_ADD      DB 0                    ;HP-HIL 'address' of mouse
0151  00                          PGID_VECT_NUM  DB 0                    ;HP_VECTOR_TABLE vector address of PGID

                                  ;JUMP TABLE FOR MS-DOS DRIVER COMMANDS

0152  02A7 R                      CMD_TABLE      DW OFFSET INIT_CODE       ;Initialize driver
0154  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Media check
0156  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Build BPB
0158  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;IOCTL input
015A  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Input
015C  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Non-destructive input
015E  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Input status
0160  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Flush input buffer
0162  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Output
0164  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Output with verify
0166  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Output status
0168  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Flush output buffer
016A  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;IOCTL output
016C  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Open device
016E  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Close device
0170  0292 R                                     DW OFFSET UNSUPPORT_CMD   ;Removeble media check

                                  ;****************  DATA AREA FOR EX-BIOS DRIVER PORTION  **************

                                  PAGE
                                  ;*********************************************************************
                                  ;*                        MOUSE DRIVER CODE                          *
                                  ;*********************************************************************

0172                              MOUSE_INT:

                                  ;PRESERVE MACHINE STATE

0172  9C                                 PUSHF                           ;Save the registers
0173  60                                 PUSHA
0174  1E                                 PUSH    DS
0175  06                                 PUSH    ES
0176  8C C8                              MOV     AX,CS                   ;Re-establish data segment addressibility
0178  8E D8                              MOV     DS,AX

                                  ;ISSUE END-OF-INTERRUPT TO 8259A

017A  B0 20                              MOV     AL,20H                  ;EOI
017C  E6 20                              OUT     20H,AL

                                  ;GET CHARACTER FROM MOUSE

017E  B8 0040                            MOV     AX,40H                  ;Get base address of COM port from table
0181  8E C0                              MOV     ES,AX
0183  2E: 8B 1E 0128 R                   MOV     BX,COM_NUMBER
0188  26: 8B 17                          MOV     DX,ES:[BX]

018B  EC                                 IN      AL,DX                   ;Get character

                                  ;STORE IN TEMPORARY BUFFER UNTIL ENTIRE FRAME HAS BEEN RECEIVED

018C  2E: 8B 1E 013A R                   MOV     BX,FRAME_COUNT          ;Get number of characters left in frame
0191  0B DB                              OR      BX,BX                   ;See if we're looking for synch byte
0193  75 0D                              JNZ     MSI_1                   ;Jump if not
0195  8A E0                              MOV     AH,AL                   ;Save a copy of mouse character
0197  24 F8                              AND     AL,0F8H                 ;Mask off button bits
0199  3C 80                              CMP     AL,80H                  ;See if this is a synch byte
019B  8A C4                              MOV     AL,AH                   ;Get the original character back
019D  74 03                              JZ      MSI_1                   ;Put character in temporary buffer if synch
                                                                         ;byte is valid
019F  E9 0260 R                          JMP     MSI_5                   ;Otherwise, throw character away

01A2  2E: 88 87 013C R            MSI_1: MOV     TEMP_BUFFER[BX],AL      ;Store character away
01A7  43                                 INC     BX                      ;Update the frame counter
01A8  2E: 89 1E 013A R                   MOV     FRAME_COUNT,BX          ;And save it
01AD  83 FB 05                           CMP     BX,5                    ;Is this the last character in frame?
01B0  74 03                              JZ      MSI_2                   ;Process the frame if so
01B2  E9 0260 R                          JMP     MSI_5                   ;Otherwise, skip on

                                  ;CHECK FOR A CHANGE IN BUTTON STATE

01B5  BB 0000                     MSI_2: MOV     BX,0                    ;New character count
01B8  2E: 89 1E 013A R                   MOV     FRAME_COUNT,BX          ;Store it

01BD  2E: 8A 87 013C R                   MOV     AL,TEMP_BUFFER[BX]      ;Get synch byte
01C2  2E: 8A 26 0141 R                   MOV     AH,LAST_SYNCH           ;Get last synch byte
01C7  2E: A2 0141 R                      MOV     LAST_SYNCH,AL           ;Update last byte
01CB  3A E0                              CMP     AH,AL                   ;See if they are the same
01CD  74 56                              JZ      MSI_3                   ;Skip on if so (no change in button state)

                                  ;SEND BUTTON ISR EVENT RECORD(S) TO INPUT SYSTEM
```

```
01CF  53                          PUSH  BX                  ;Save frame counter.
01D0  52                          PUSH  DX                  ;Save
01D1  32 E0                       XOR   AH,AL               ;AH now holds mask of buttons that have changed.
01D3  B7 01                       MOV   BH,01H              ;Mask for first button
01D5  B9 0003                     MOV   CX,3                ;Number of buttons to process.

01D8                  MBUTTON:
01D8  8A DC                       MOV   BL,AH               ;Get a copy of change mask
01DA  22 DF                       AND   BL,BH               ;See if selected button was the one that changed.
01DC  74 41                       JZ    MNEXT_BUTTON        ;Skip on if not
01DE  84 F8                       TEST  BH,AL               ;Determine state (make or break) of selected butt

01E0  74 04                       JZ    MBUTTON_DOWN

01E2                  MBUTTON_UP:

01E2  B3 80                       MOV   BL,80H              ;Set bit 7 (make/break bit) to 0 (break).
01E4  EB 02                       JMP   SHORT MBUTTON_ISR

01E6                  MBUTTON_DOWN:

01E6  B3 00                       MOV   BL,00H              ;Set bit 7 (make/break bit) to 1 (make).

01E8                  MBUTTON_ISR:
01E8  53                          PUSH  BX
01E9  8B D9                       MOV   BX, CX
01EB  32 FF                       XOR   BH, BH
01ED  FE CB                       DEC   BL
01EF  2E: 8A 8F 01F7 R            MOV   CL, CS:BUTTON_TAB[BX]
01F4  5B                          POP   BX
01F5  EB 03                       JMP   SHORT BISR2
01F7  00              BUTTON_TAB  DB    0                   ; left button
01F8  02                          DB    2                   ; middle button
01F9  01                          DB    1                   ; right button
01FA                  BISR2:
01FA  0A D9                       OR    BL, CL
01FC  32 FF                       XOR   BH, BH              ; clear out bh

01FE  50                          PUSH  AX                  ;Save registers
01FF  53                          PUSH  BX
0200  51                          PUSH  CX
0201  1E                          PUSH  DS

                                  page
                      ;Create ISR Event Record

0202  B6 09                       MOV   DH,T_KC_BUTTON      ;Set data type.
0204  2E: 8A 16 0151 R            MOV   DL,PGID_VECT_NUM    ;Get vector number of mouse's PGID
0209  B9 0000                     MOV   CX,0                ;Burst length (N/A)
020C  8C C8                       MOV   AX,CS               ;Point ES:0 to driver header.
020E  40                          INC   AX
020F  40                          INC   AX
0210  8E C0                       MOV   ES,AX
0212  B4 00                       MOV   AH,F_ISR            ;Set ISR function
0214  BD 002A                     MOV   BP,V_SINPUT         ;We're calling the INPUT driver
0217  FA                          CLI                       ;Turn off interrupts while we're out
0218  CD 6F                       INT   HP_ENTRY
021A  FB                          STI                       ;Re-enable interrupts.

021B  1F                          POP   DS
021C  59                          POP   CX
021D  5B                          POP   BX
021E  58                          POP   AX

021F                  MNEXT_BUTTON:

021F  D0 E7                       SHL   BH,1                ;Move button selector mask to next button.
0221  E2 B5                       LOOP  MBUTTON
0223  5A                          POP   DX                  ;Restore
0224  5B                          POP   BX                  ;Get frame counter back.

                      ;CHECK FOR MOTION

0225  43              MSI_3:      INC   BX                  ;Point to first delta X in buffer.
0226  2E: 8A 97 013C R            MOV   DL,TEMP_BUFFER[BX]
022B  43                          INC   BX                  ;Get first delta Y.
022C  2E: 8A B7 013C R            MOV   DH,TEMP_BUFFER[BX]
0231  43                          INC   BX                  ;Add second delta X to first.
0232  2E: 02 97 013C R            ADD   DL,TEMP_BUFFER[BX]
0237  43                          INC   BX                  ;Add second delta Y to first.
0238  2E: 02 B7 013C R            ADD   DH,TEMP_BUFFER[BX]

023D  0B D2           MSI_4:      OR    DX,DX               ;Check for zero motion.
023F  74 1F                       JZ    MSI_5               ;Skip on if none detected.

                                  page
                      ;SEND MOTION ISR EVENT RECORD TO INPUT SYSTEM

0241  8A C2                       MOV   AL,DL               ;Convert delta X to 16 bit value and put
0243  98                          CBW                       ;it in ISR Event Record (BX register).
0244  8B D8                       MOV   BX,AX
0246  8A C6                       MOV   AL,DH               ;Ditto for delta Y (CX register).
0248  98                          CBW
0249  8B C8                       MOV   CX,AX
```

RS-232 Mouse Driver

```
                                  ;Create motion ISR event record
0248  B6 41                              MOV    DH,T_REL16            ;Set ISR Event record data type to 16 bit
                                                                      ;relative motion
024D  2E  8A 16 0151 R                   MOV    DL,PGID_VECT_NUM      ;Get vector number of mouse's PGID
0252  8C C8                              MOV    AX,CS                 ;Set ES:0 to driver header.
0254  40                                 INC    AX
0255  8E C0                              MOV    ES,AX
0257  B4 00                              MOV    AH,F_ISR              ;Select ISR function
0259  BD 002A                            MOV    BP,V_SINPUT           ;We're passing this on to the INPUT driver.
025C  FA                                 CLI                          ;Interrupts are supposed to be off.
025D  CD 6F                              INT    HP_ENTRY
025F  FB                                 STI                          ;Turn interrupts back on now.

                                  ;RESTORE MACHINE STATE AND EXIT

0260  07                          MSI_5: POP    ES
0261  1F                                 POP    DS
0262  61                                 POPA
0263  9D                                 POPF
0264  CF                                 IRET

                                  PAGE

                                  ;********************************************************************
                                  ;*                        MS-DOS DRIVER CODE                        *
                                  ;********************************************************************

                                  ;**********************  STRATEGY ENTRY POINT  **********************

0265                              DEV_STRATEGY PROC FAR
0265  2E  89 1E 0050 R                   MOV    CS:REQ_HDR_OFF,BX     ;Save offset of request header.
026A  2E  8C 06 0052 R                   MOV    CS:REQ_HDR_SEG,ES     ;Save segment of request header.
026F  CB                                 RET                          ;Return to MS-DOS.
0270                              DEV_STRATEGY ENDP

                                  ;**********************  INTERRUPT ENTRY POINT  *********************

0270                              DEV_INTERRUPT PROC FAR

                                  ;SAVE MACHINE STATE
0270  9C                                 PUSHF
0271  FC                                 CLD
0272  60                                 PUSHA                        ;Save registers.
0273  8C CF                              MOV    DI,CS                 ;Set DS to CS.
0275  8E DF                              MOV    DS,DI

                                  ;FETCH COMMAND FROM REQUEST HEADER
0277  2E  C4 3E 0050 R                   LES    DI,DWORD PTR REQ_HDR_OFF ;Move address of request header into ES:DI.
027C  26  8A 45 02                       MOV    AL,ES:[DI].RH_CMD_CODE   ;Get command byte from header
0280  3C 00                              CMP    AL,MSD_INIT              ;Perform range check on command byte.
0282  72 0E                              JB     BAD_CMD
0284  3C 0F                              CMP    AL,MSD_REM_MEDIA
0286  77 0A                              JA     BAD_CMD
0288  98                                 CBW                          ;Convert command into jump table offset
0289  D1 E0                              SHL    AX,1
028B  8B D8                              MOV    BX,AX
028D  2E  FF A7 0152 R                   JMP    CMD_TABLE[BX]         ;Dispatch to requested function.

                                  ;EXIT POINT FOR BAD OR UNSUPPORTED FUNCTIONS

0292                              BAD_CMD:
0292                              UNSUPPORT_CMD:

0292  26  81 4D 03 8000                  OR     ES:[DI].RH_STATUS, MASK ERROR ;Set error flag in return status word.
0298  26  81 4D 03 0003                  OR     ES:[DI].RH_STATUS, MSD_UNKNOWN_CMD ;Set error code.

                                  ;COMMON EXIT POINT

029E  26  81 4D 03 0100           EXIT:  OR     ES:[DI].RH_STATUS, MASK DONE ;Set return status to done.
02A4  61                                 POPA                         ;Restore registers.
02A5  9D                                 POPF                         ;Restore flags.
02A6  CB                                 RET                          ;Return to MS-DOS.


                                  ;**********************  END OF RESIDENT CODE  **********************
                                  PAGE
                                  ;**********************  INITIALIZATION CODE  ***********************

02A7                              INIT_CODE:

                                  ;SET UP LOCAL STACK
02A7  FA                                 CLI                          ;Disable interrupts while we're messing with stack.

02A8  BE 0124 R                          MOV    SI,OFFSET STACK_PTR   ;Store existing stack environment.
02AB  89 24                              MOV    [SI],SP
02AD  83 C6 02                           ADD    SI,2
02B0  8C 14                              MOV    [SI],SS

02B2  BC 0511 R                          MOV    SP,OFFSET CS:STACK_TOP ;Set up our local stack.
02B5  8C C8                              MOV    AX,CS                 ;Stack segment is same as code (CS)
02B7  8E D0                              MOV    SS,AX

02B9  FB                                 STI                          ;Re-enable interrupts.
```

```
                                        ;PRINT SIGN-ON MESSAGE

02BA  BA 0054 R                                 MOV     DX,OFFSET SIGN_ON_MSG
02BD  B4 09                                     MOV     AH,PRINT_STR
02BF  CD 21                                     INT     DOS_ENTRY

                                        ;PARSE CONFIG.SYS COMMAND LINE TO DETERMINE WHICH COM PORT THE MOUSE IS ON

02C1  BB 0000                                   MOV     BX,0                    ;Clear BX. It will be used as index into
                                                                                ;command line
02C4  26  C4 7D 12                              LES     DI,ES:[DI].RH_CMD_LINE  ;Load ES:DI with pointer to CONFIG.SYS command
                                                                                ;line

02C8  26  8A 01                         IC_1:   MOV     AL,BYTE PTR ES:[DI+BX]  ;Get next charactor in command line
02CB  3C 2F                                     CMP     AL,'/'                  ;Check for backslash
02CD  74 0B                                     JZ      IC_2                    ;If found, indicates start of parameters
02CF  3C 0D                                     CMP     AL,CR                   ;Check for carriage return (Indicates a bogas
                                                                                ;set of parameters)
02D1  74 1C                                     JZ      IC_3                    ;If found, stop scanning command line
02D3  3C 0A                                     CMP     AL,LF                   ;Check for line feed (Indicates no parameters
                                                                                ;entered in command line
02D5  74 18                                     JZ      IC_3                    ;If found, stop scanning command line
02D7  43                                        INC     BX                      ;Else, point to next character,
02D8  EB EE                                     JMP     IC_1                    ;and continue scanning command line

02DA  43                                IC_2:   INC     BX                      ;Get next character. Should indicate COM port
02DB  26  8A 01                                 MOV     AL,BYTE PTR ES:[DI+BX]  ;to use. Valid range is 1 - 4
02DE  2C 31                                     SUB     AL,'1'                  ;Convert number into offset from 1
02E0  72 0D                                     JB      IC_3                    ;Perform range check on results
02E2  3C 03                                     CMP     AL,3
02E4  77 09                                     JA      IC_3
02E6  98                                        CBW                             ;Convert into offset into STD-BIOS COM port
                                                                                ;base address table at 0040:0000H
02E7  D1 E0                                     SHL     AX,1
02E9  2E  A3 0128 R                             MOV     COM_NUMBER,AX           ;Save it for future use
02ED  EB 07                                     JMP     SHORT IC_4

02EF  2E  C7 06 0128 R 0000             IC_3:   MOV     COM_NUMBER,0            ;If we wind up here, there were no parameters
                                                                                ;specified in the command line, or an invalid
                                                                                ;COM port was specified. Set COM port COM1
                                                                                ;default.

02F6  8B D8                             IC_4:   MOV     BX,AX                   ;Convert offset into ASCII COM number (1 - 4)
02F8  D1 EB                                     SHR     BX,1
02FA  80 C3 31                                  ADD     BL,'1'
02FD  2E  88 1E 00C1 R                          MOV     COM_MSG,BL              ;Store in sign-on message
0302  FA                                        CLI                             ;Disable interrupts while mouse interrupt
                                                                                ;is being set up.

                                        ;INITIALIZE SERIAL PORT PARAMETERS

0303  8B F8                                     MOV     DI,AX                   ;Move COM port table offset into DI
0305  B8 0040                                   MOV     AX,40H                  ;Segment address of COM port base address table
0308  8E C0                                     MOV     ES,AX
030A  26  8B 15                                 MOV     DX,ES:[DI]              ;Get base address of COM port out of table
030D  0B D2                                     OR      DX,DX                   ;Make sure port exists
030F  75 03                                     JNZ     IC_4A                   ;Continue with initialization if it does,
0311  E9 03B2 R                                 JMP     INIT_NO_PORT            ;otherwise, go to error routine.

                                        ;Clear existing error or character

0314  83 C2 05                          IC_4A:  ADD     DX,5
0317  EC                                        IN      AL,DX
0318  EB 00                                     JMP     SHORT $+2

                                        ;Set baud rate divisor to 1200 baud

031A  83 EA 02                                  SUB     DX,2                    ;Point to line control register
031D  B0 80                                     MOV     AL,80H                  ;Set line control register to divisor programming
031F  EE                                        OUT     DX,AL                   ;mode
0320  EB 00                                     JMP     SHORT $+2               ;Delay
0322  83 EA 03                                  SUB     DX,3                    ;Point to divisor LSB register (base).
0325  B0 60                                     MOV     AL,60H                  ;LSB for 1200 bps.
0327  EE                                        OUT     DX,AL
0328  EB 00                                     JMP     SHORT $+2               ;Delay.
032A  42                                        INC     DX                      ;Point to MSB of divisor (base + 1).
032B  B0 00                                     MOV     AL,00H                  ;MSB for 1200 bps.
032D  EE                                        OUT     DX,AL
032E  EB 00                                     JMP     SHORT $+2               ;Delay.

                                        ;Initialize line control register.

0330  83 C2 02                                  ADD     DX,2                    ;Point to line control register (base +3).
0333  B0 03                                     MOV     AL,03H                  ;8 data bits, 1 stop bit, no parity
0335  EE                                        OUT     DX,AL
0336  EB 00                                     JMP     SHORT $+2               ;Delay.

                                        ;Initialize modem control register

0338  42                                        INC     DX                      ;Point to modem control register (base + 4).
0339  B0 0B                                     MOV     AL,0BH                  ;DTR and RTS set, OUT2 set to enable interrupts.
```

RS-232 Mouse Driver

```
      033B  EE                               OUT    DX,AL
      033C  EB 00                            JMP    SHORT $+2              ;Delay

                                     ;Initialize interrupt enable register

      033E  83 EA 03                         SUB    DX,3                   ;Point to interrupt enable register (base + 1)
      0341  B0 01                            MOV    AL,01                  ;Enable Rx Data Ready interrupt
      0343  EE                               OUT    DX,AL

                                     ;SET UP COM PORT INTERRUPT VECTOR

      0344  2E: 8B 1E 0128 R                 MOV    BX,COM_NUMBER          ;Get table offset back
      0349  2E: 8B BF 012A R                 MOV    DI,INT_TABLE[BX]       ;Use it as index into interrupt vector table
      034E  B8 0000                          MOV    AX,0                   ;Set ES to interrupt vector segment (0 )
      0351  8E C0                            MOV    ES,AX
      0353  B8 0172 R                        MOV    AX,OFFSET MOUSE_INT    ;Initialize vector
      0356  AB                               STOSW
      0357  8C C8                            MOV    AX,CS
      0359  AB                               STOSW

                                     ;ENABLE MOUSE INTERRUPT ON 8259A INTERRUPT CONTROLLER

      035A  2E: 8B 8F 0132 R                 MOV    CX,MASK_TABLE[BX]      ;Get mask from table
      035F  E4 21                            IN     AL,21H                 ;Get current mask
      0361  EB 00                            JMP    SHORT IC_10            ;Delay
      0363  22 C1                    IC_10:  AND    AL,CL                  ;Clear mask for mouse interrupt
      0365  E6 21                            OUT    21H,AL                 ;Set new value

      0367  FB                               STI                           ;Re-enable interrupts

      0368  B4 0C                            MOV    AH,F_INQUIRE_ENTRY     ;Return CS IP of PGID driver function
      036A  BD 002A                          MOV    BP,V_SINPUT
      036D  1E                               PUSH   DS
      036E  CD 6F                            INT    HP_ENTRY
      0370  1F                               POP    DS
      0371  80 FC 02                         CMP    AH,RS_UNSUPPORTED      ;See if brute force approach is necessary
      0374  75 06                            JNE    INIT_3
      0376  0E                               PUSH   CS                     ;Even the best laid plans of mice and men aft
      0377  07                               POP    ES                     ;go awry  F_INQUIRE_PGID is not implemented in
      0378  8D 1E 03FF R                     LEA    BX, CS:PGID_DRIVER     ;some early ROM versions  The PGID CS IP must be
                                                                           ;hard coded for these systems

      037C  8B FB                    INIT_3: MOV    DI,BX                  ;Move IP into DI
      037E  8C CA                            MOV    DX,CS                  ;Get PGID's DS
      0380  83 C2 02                         ADD    DX, 2                  ;account for ORG 20H
      0383  B4 0A                            MOV    AH,F_INS_XCHGFREE      ;Exchange fixed vector address function
      0385  BD 0012                          MOV    BP,V_SYSTEM
      0388  1E                               PUSH   DS
      0389  CD 6F                            INT    HP_ENTRY
      038B  1F                               POP    DS
      038C  80 FC F6                         CMP    AH, RS_NO_VECTOR       ;Is it installed in vector table
      038F  74 18                            JE     INIT_NO_VECTOR

      0391  8B C3                            MOV    AX, BX                 ;Set up for the divide
      0393  B3 06                            MOV    BL, 6
      0395  F6 F3                            DIV    BL                     ;Convert to a vector index
      0397  2E: A2 0151 R                    MOV    PGID_VECT_NUM, AL      ;Save for ISR Events

      039B  B4 04                            MOV    AH, F_IO_CONTROL       ; Now to make sure that the V_LHPMOUSE
      039D  B0 02                            MOV    AL, SF_MOUSE_OVERRIDE  ; driver sets up INT 33H
      039F  BD 00CC                          MOV    BP, V_LHPMOUSE
      03A2  1E                               PUSH   DS
      03A3  CD 6F                            INT    HP_ENTRY
      03A5  1F                               POP    DS

      03A6  EB 13 90                         JMP    INIT_OK

      03A9                           INIT_NO_VECTOR:

      03A9  BA 0103 R                        MOV    DX,OFFSET NO_VECTOR    ;Print error message
      03AC  B4 09                            MOV    AH,PRINT_STR
      03AE  CD 21                            INT    DOS_ENTRY
      03B0  EB 14                            JMP    SHORT INIT_EXIT

      03B2                           INIT_NO_PORT:

      03B2  BA 00C8 R                        MOV    DX,OFFSET NO_PORT_MSG  ;Print error message
      03B5  B4 09                            MOV    AH,PRINT_STR
      03B7  CD 21                            INT    DOS_ENTRY
      03B9  EB 08                            JMP    SHORT INIT_EXIT

      03BB                           INIT_OK:

      03BB  8C C8                            MOV    AX,CS                  ;Set DS back to proper value
      03BD  8E D8                            MOV    DS,AX
      03BF  BA 00AB R                        MOV    DX,OFFSET OK_MSG       ;Print sign-on message
      03C2  B4 09                            MOV    AH,PRINT_STR           ;MS-DOS print string function number
      03C4  CD 21                            INT    DOS_ENTRY

      03C6                           INIT_EXIT:

      03C6  06                               PUSH   ES                     ; now to set the number of buttons
      03C7  50                               PUSH   AX                     ; V_LHPMOUSE has
```

```
03C8  B8 0000                      MOV     AX, 0
03CB  8E C0                        MOV     ES, AX
03CD  26  8E 06 01BE               MOV     ES, ES:[HP_ENTRY * 4 + 2]
03D2  26  8E 06 00D0               MOV     ES, ES:[V_LHPMOUSE+4]
03D7  26  C6 06 004C 03            MOV     BYTE PTR ES:MSE_NUM_BUTTON,3   ;Define the number of buttons to 3
03DD  58                           POP     AX
03DE  07                           POP     ES

03DF  2E  C4 3E 0050 R             LES     DI,DWORD PTR REQ_HDR_OFF  ;Reload ES:DI with address of request header
03E4  26: C7 45 0E 04D1 R          MOV     ES:[DI].RH_END_OFF,OFFSET END_OF_DRIVER ;Return end of resident code to
03EA  26: 8C 4D 10                 MOV     ES:[DI].RH_END_SEG,CS              ;MS-DOS

                            ;RESTORE OLD STACK FRAME AND EXIT

03EE  FA                           CLI                              ;Disable interrupts while working on stack frame
03EF  BE 0124 R                    MOV     SI,OFFSET STACK_PTR      ;Get address of old stack storage
03F2  8B 24                        MOV     SP,[SI]                  ;Restore stack pointer
03F4  83 C6 02                     ADD     SI,2                     ;Get old stack segment
03F7  8B 04                        MOV     AX,[SI]                  ;And restore it
03F9  8E D0                        MOV     SS,AX
03FB  FB                           STI                              ;Re-enable interrupts

03FC  E9 029E R                    JMP     EXIT

03FF                        DEV_INTERRUPT  ENDP
03FF                        DEV_DRIVER     ENDP

                                           page
                                           list
                            ;***DRIVER HEADER*************************************************
                            ;
                            ;  NAME: PGID_DRIVER
                            ;
                            ;  DESCRIPTION:
                            ;
                            ;  LIST OF FUNCTIONS:   (function code in hex)
                            ;    [Those functions not listed are NOT_SUPPORTED.]
                            ;
                            ;        F_ISR
                            ;        F_SYSTEM
                            ;
                            ;  PARAMETERS:
                            ;        See function headers for specific values for other entry and exit
                            ;        parameters
                            ;
                            ; REGISTERS PRESERVED:
                            ;
                            ;  DEFINITION MODIFICATION HISTORY
                            ;
                            ;    VERSION:
                            ;
                            ;    DESCRIPTION OF CHANGES:
                            ;
                            ;*****************************************************************

                                           subttl PGID Main entry point

                                           page
                                           assume  cs:CODE, ds:nothing
                                           public  PGID_DRIVER
                            ;
                            ; NOTE **** No driver header for PGID ****
                            ;  Only 2 functions are supported: F_ISR, F_SYSTEM -- all others are unsupported


03FF                        pgid_driver    proc    near
03FF  80 FC 00                             cmp     ah,F_ISR                 ; F_ISR?
0402  75 04                                jne     check_f_system
0404  E8 0414 R                            call    pgid_Isr
0407  CF                                   iret

0408                        check_f_system:
0408  80 FC 02                             cmp     ah,F_SYSTEM              ; F_SYSTEM?
040B  75 04                                jne     pgid_opcode_bad
040D  E8 0498 R                            call    pgid_system
0410  CF                                   iret                             ; function has set return code

                            ;
                            ;-----------------------------------------------------
                            ; Main opcode out of range of PGID functions supported
                            ;   just return RS_UNSUPPORTED
                            ;-----------------------------------------------------
0411                        pgid_opcode_bad:
0411  B4 02                                mov     ah,RS_UNSUPPORTED
0413  CF                                   iret
                            ;
0414                        pgid_driver    endp
                                           page
                            ;***FUNCTION HEADER***********************************************
                            ;
                            ;  NAME:  PGID_ISR
                            ;
                            ;  FUNCTIONAL DESCRIPTION:
```

RS-232 Mouse Driver

```
                                  ;       A graphics input device (GID) physical event has occurred which
                                  ;  caused an F_ISR request.  If the event was a button press, then
                                  ;  the D_STATE and D_TRANSITION fields will be adjusted and the parent
                                  ;  driver will be called immediately.
                                  ;       If a the event was a movement, this function will update the
                                  ;  absolute position field if the device is a relative device or will
                                  ;  update the relative position field if it's an absolute device.  It
                                  ;  will then call the PARENT driver to handle the movement event.
                                  ;**************************************
                                  ;  NOTE: The PGID driver takes HP-HIL 'Y' axis data and translates
                                  ;        it into INDUSTRY-STANDARD space data (flips the Y axis).
                                  ;        HP-HIL has positive 'Y' in the upward direction, while
                                  ;        INDUSTRY-STD  is downward.
                                  ;**************************************
                                  ;
                                  ;  PARAMETERS:
                                  ;
                                  ;    ON ENTRY:
                                  ;       AH = F_ISR
                                  ;       DH = D_TYPE
                                  ;       DL = SOURCE  Vector Index
                                  ;       DS:0  = Pointer to Physical device header and describe record
                                  ;  For Button Event (Keycode Event Record)   (D_TYPE = T_KC_BUTTON)
                                  ;       BX =  Button transition information
                                  ;               bits 0-6:  buttons
                                  ;               bits 7:    0  up transition
                                  ;                          1  down transition
                                  ; For Movement Event (GID Event Record, D_TYPE = T_REL08, T_REL16,
                                  ;                                                T_ABS08, or T_ABS16)
                                  ;       BX =  AXIS-0 (X) Movement in RAW data form (SIGN EXTENDED, if necessary)
                                  ;       CX =  AXIS-1 (Y) Movement in RAW data form (SIGN EXTENDED, if necessary)
                                  ;
                                  ;    ON EXIT:
                                  ;       AH = Return Code (SET BY PARENT Driver)
                                  ;
                                  ;  REGISTERS ALTERED: ax,bx,cx
                                  ;
                                  ;  DEFINITION MODIFICATION HISTORY:
                                  ;
                                  ;    VERSION:
                                  ;
                                  ;    DESCRIPTION OF CHANGES:
                                  ;
                                  ;*****************************************************************
                                                  page
      0414                        pgid_isr        proc    near
                                  ;--------------------------------
                                  ; See if this was a button event
                                  ;--------------------------------
      0414  80 FE 09                              cmp     dh,T_KC_BUTTON          ; D_TYPE = T_KC_BUTTON ?
      0417  74 57                                 je      short button_isr        ; adjust D_STATE & D_TRANSITION
                                  ;
                                  ;--------------------------------------------------
                                  ; A movement occurred.  If this was an absolute device
                                  ;   that moved, then adjust the relative location field in the describe record.
                                  ;   If it was a relative device, then adjust the absolute location field
                                  ;   in the describe record.
                                  ;  BX,CX have X,Y movement respectively.
                                  ;--------------------------------------------------
      0419                        movement_isr:
      0419  80 FE 40                              cmp     dh,T_REL08              ; relative 8 bit movement
      041C  74 3E                                 je      short rel_move
      041E  80 FE 41                              cmp     dh,T_REL16              ; relative 16 bit movement
      0421  74 39                                 je      short rel_move
      0423  80 FE 42                              cmp     dh,T_ABS08              ; absolute 8 bit movement
      0426  74 08                                 je      short abs_move
      0428  80 FE 43                              cmp     dh,T_ABS16              ; absolute 16 bit movement
      042B  74 03                                 je      short abs_move
                                  ;
                                  ;---------------------------------------
                                  ; If none of the above devices, then this is a bad input device
                                  ;---------------------------------------
      042D  B4 FE                                 mov     ah,RS_FAIL              ; return RS_FAIL
      042F  C3                                    ret                             ; return to main driver
                                                  page
                                  ;**************************
                                  ; Absolute movement
                                  ;--------------------------
                                  ; We must invert the Y axis to put into INDUSTRY STANDARD coordinate space.
                                  ; Must convert 'Y' coordinate such that negative movement is upward (opposite
                                  ;        of HP-HIL definition.)
                                  ; -- Set BX,CX (x,y ABSOLUTE movement) for event record when done, then pass
                                  ;                 event record to parent driver.
                                  ;
                                  ; (BX) is 'X' HP-HIL coordinate
                                  ; (CX) is 'Y'   [ ABS_Y(std) = D_SIZE_Y - ABS_Y(hphil) ]
                                  ;**************************
      0430                        abs_move:
      0430  87 1E 0014                            xchg    bx,ds:D_ABS_X   ; save new x position
      0434  2B 1E 0014                            sub     bx,ds:D_ABS_X   ; (OLD - NEW)
      0438  F7 DB                                 neg     bx              ; Relative move = (NEW - OLD)
      043A  89 1E 0018                            mov     ds:D_REL_X,bx   ; save new x relative
```

```
        043E  8B 1E 0012                        mov     bx,ds:D_SIZE_Y    ; 'Y' limit
        0442  2B D9                             sub     bx,cx             ; invert the axis: bx = (LIMIT - y)
        0444  87 1E 0016                        xchg    bx,ds:D_ABS_Y     ; New ABS Y
        0448  2B 1E 0016                        sub     bx,ds:D_ABS_Y     ; (OLD - NEW)
        044C  F7 DB                             neg     bx                ; Relative move = (NEW - OLD)
        044E  89 0E 0018                        mov     ds:D_REL_X,cx     ; save new Y relative

                                ;-----------------------------------------------
                                ; GET the X,Y absolute coordinates for the event record
                                ;-----------------------------------------------
        0452  8B 1E 0014                        mov     bx,ds:D_ABS_X
        0456  8B 0E 0016                        mov     cx,ds:D_ABS_Y
        045A  EB 31                             jmp     short give_to_parent     ; ok to pass event to parent
                                                page
                                ;*****************************
                                ; Relative movement
                                ;-----------------------------
                                ; We must invert the Y axis to put into INDUSTRY STANDARD coordinate space.
                                ; Must convert 'Y' coordinate such that negative movement is upward (opposite
                                ;       of HP-HIL definition.)
                                ; -- Set BX,CX (x,y RELATIVE movement) for event record when done, then pass
                                ;                       event record to parent driver
                                ;
                                ; (BX) is 'X' HP-HIL coordinate.
                                ; (CX) is 'Y'   [ REL_Y(std) = -REL_Y(hphil) ]
                                ;*****************************
        045C                    rel_move:
        045C  89 1E 0018                        mov     ds:D_REL_X,bx     ; save new rel. move (X)
        0460  F7 D9                             neg     cx                ; CONVERT TO INDUSTRY STD. SPACE
        0462  89 0E 001A                        mov     ds:D_REL_Y,cx     ; save new rel. move (Y)

        0466  01 1E 0014                        add     ds:D_ABS_X,bx     ; add new X relative movement
        046A  01 0E 0016                        add     ds:D_ABS_Y,cx     ; add new Y relative movement

                                ;-----------------------------------------------
                                ;  BX,CX still contain X,Y relative movement information for the event record
                                ;-----------------------------------------------
        046E  EB 1D                             jmp     short give_to_parent     ; ok to pass event to parent
                                                page
                                ;-------------------------
                                ; Button Press/Release ISR
                                ;   Adjust the D_TRANSITION and D_STATE fields of the physical device's
                                ;   describe record
                                ;
                                ;     Assuming  1. Only one button can make a transition at a time.
                                ;               2. The button only either goes up or down, not both
                                ;               3. No strings of buttons are sent (CX register available)
                                ;
                                ; BL is number of button that changed
                                ;       bit 7 is the up/down (1/0) flag
                                ;-------------------------
        = 0080                  UP_DOWN_BIT     equu    10000000B         ; bit 7 is up (1), down(0) bit

        0470                    button_isr
                                ;---------------------------------------
                                ; Convert button number to bit mask corresponding
                                ;   to the changed button
                                ;---------------------------------------
        0470  8A CB                             mov     cl,bl             ; get button # keycode in CL for shift
        0472  80 E1 7F                          and     cl,01111111B      ; keep button #, get rid of up/down flag
        0475  B0 01                             mov     al,00000001B      ; put '1' in bit 0 of al
        0477  D2 E0                             shl     al,cl             ; set appropriate button bit mask

        0479  A2 000C                           mov     ds:D_TRANSITION,al        ; note which button changed

        047C  F6 C3 80                          test    bl,UP_DOWN_BIT    ; [bit 7] Was it UP = 1 or down = 0
        047F  74 08                             jz      short button_down
        0481                    button_up:
        0481  08 06 000D                        or      ds:D_STATE,al     ; set the button = 1 (up)
        0485  EB 06                             jmp     short give_to_parent     ; ok to pass event to parent
        0487                    button_down
        0487  F6 D0                             not     al                ; invert for clearing the bit
        0489  20 06 000D                        and     ds:D_STATE,al     ; clear the button to 0 (down)

                                ;  fall through to GIVE_TO_PARENT code -- ok to pass event to parent now
                                ;               jmp     give_to_parent    ; (COMMENTED OUT -- jump not necessary)
                                                page
                                ; ok to pass event to parent now
                                ;---------------------------------------
                                ; Call PARENT driver to handle the ISR
                                ; NOTE: HPHIL driver has already adjusted D_SOURCE field, HPHIL_ID and other
                                ;       relevant HPHIL info before passing the event up to here.
                                ;---------------------------------------
        048D                    give_to_parent
        048D  B4 00                             mov     ah,F_ISR                  ; tell parent: ISR function
        048F  8B 2E 000A                        mov     bp,ds:DH_V_PARENT         ; parent's vector
        0493  CD 6F                             INT     HP_ENTRY
        0495  C3                                ret                               ; return to main driver
        0496                    pgid_isr        endp

                                                subttl  PGID_SYSTEM function
```

```
                                                page
                        ;***FUNCTION HEADER*************************************************
                        ;
                        ;  NAME    PGID_SYSTEM
                        ;
                        ;  FUNCTIONAL DESCRIPTION
                        ;
                        ;       This function supports the HP SYSTEM subfunctions requested of
                        ;  the PGID driver.  The subfunction is checked to make sure that it
                        ;  is in the appropriate range.
                        ;
                        ;
                        ;  PARAMETERS
                        ;
                        ;    ON ENTRY
                        ;       AH = F_SYSTEM
                        ;       AL = SYSTEM subfunction code
                        ;
                        ;         F_SYSTEM Subfunctions (in hex):
                        ;             (functions not included are UNSUPPORTED)
                        ;         SF_INIT
                        ;         SF_START
                        ;         SF_REPORT_STATE
                        ;         SF_VERSION_DESC
                        ;
                        ;    ON EXIT
                        ;       See individual system subfunctions for values returned.
                        ;       RS_UNSUPPORTED will be returned if the subfunction is out of range.
                        ;
                        ;  REGISTERS PRESERVED:
                        ;
                        ;  DEFINITION MODIFICATION HISTORY
                        ;
                        ;    VERSION:
                        ;
                        ;    DESCRIPTION OF CHANGES:
                        ;
                        ;*******************************************************************
                                        page
 0496                   pgid_system     proc    near

 0496  3C 06 90 90                      cmp     al,MAX_PGID_SYS_FN      ; check bounds
 049A  77 0D                            ja      short pgid_sys_bad      ; out of range ?

 049C  87 EB                            xchg    bp,bx           ; save bx, set bp=subfunction code (al)
 049E  8A D8                            mov     bl,al
 04A0  32 FF                            xor     bh,bh
 04A2  87 EB                            xchg    bp,bx

 04A4  2E: FF A6 04AC R                 jmp     cs:word ptr pgid_sys_case[bp]
 04A9                   pgid_sys_bad:
 04A9  B4 02                            mov     ah,RS_UNSUPPORTED       ; bad subfunction code
 04AB  C3                               ret                             ; return to main driver

                        ;-----------------------------------
                        ; PGID_SYSTEM subfunction jump table
                        ;-----------------------------------
 04AC                   pgid_sys_case:
 04AC  04B4 R                           dw      word ptr  pgid_init     ; SF_INIT
 04AE  04BC R                           dw      word ptr  pgid_start    ; SF_START
 04B0  04BF R                           dw      word ptr  pgid_state    ; SF_REPORT_STATE
 04B2  04C2 R                           dw      word ptr  pgid_version  ; SF_VERSION_DESC
 = 0006                 MAX_PGID_SYS_FN equ     byte ptr ($ - pgid_sys_case - 2) ; max supported sys fn.
 04B4                   pgid_system     endp

                                        subttl  PGID_INIT System Subfunction
                                        page
                        ;***FUNCTION HEADER*************************************************
                        ;
                        ;  NAME:           PGID_INIT
                        ;
                        ;  FUNCTIONAL DESCRIPTION
                        ;
                        ;       System subfunction SF_INIT -- initialize the physical device
                        ;  header and describe record.  IT IS ASSUMED THAT THE HPHIL DRIVER HAS
                        ;  INITIALIZED ALL APPROPRIATE INFO ALREADY.  All position and button
                        ;  data is zeroed out, and relevant HPHIL info is already filled in.
                        ;  Only must set default button states (all off (=1)).
                        ;
                        ;  PARAMETERS
                        ;
                        ;    ON ENTRY
                        ;       AH = F_SYSTEM
                        ;       AL = SF_INIT
                        ;
                        ;    ON EXIT
                        ;       AH = Return status (RS_SUCCESSFUL)
                        ;
                        ;  REGISTERS ALTERED: ax
                        ;
                        ;  DEFINITION MODIFICATION HISTORY
                        ;
                        ;    VERSION:
```

RS-232 Mouse Driver

```
                              ;   DESCRIPTION OF CHANGES:
                              ;
                              ;*****************************************************************
 04B4                         pgid_init          proc     near
 = 00FF                       INIT_BUTTON_STATE  equ      OFFh                   ; all buttons open
                              ;
 04B4  C6 06 000D FF                             mov      ds:D_STATE,INIT_BUTTON_STATE   ; all off

 04B9  B4 00                                     mov      ah,RS_SUCCESSFUL; successful initialization
 04BB  C3                                        ret                        ; return to main driver
 04BC                         pgid_init          endp
                                                 subttl   PGID_START System Subfunction
                                                 page
                              ;***FUNCTION HEADER**********************************************
                              ;
                              ; NAME:          PGID_START
                              ;
                              ; FUNCTIONAL DESCRIPTION:
                              ;
                              ;      System subfunction SF_START -- start the driver.  This does nothing
                              ; but return with RS_SUCCESSFUL.
                              ;
                              ; PARAMETERS
                              ;
                              ;   ON ENTRY:
                              ;      AH = F_SYSTEM
                              ;      AL = SF_START
                              ;
                              ;   ON EXIT:
                              ;      AH = return status (RS_SUCCESSFUL)
                              ;
                              ; REGISTERS ALTERED: ah
                              ;
                              ; DEFINITION MODIFICATION HISTORY
                              ;
                              ;   VERSION:
                              ;
                              ;   DESCRIPTION OF CHANGES:
                              ;
                              ;*****************************************************************
 04BC                         pgid_start         proc     near
                              ;
 04BC  B4 00                                     mov      ah,RS_SUCCESSFUL; successful start up
 04BE  C3                                        ret                        ; return to main driver
 04BF                         pgid_start         endp
                                                 subttl   PGID_STATE System Subfunction
                                                 page
                              ;***FUNCTION HEADER**********************************************
                              ;
                              ; NAME:          PGID_STATE
                              ;
                              ; FUNCTIONAL DESCRIPTION
                              ;
                              ;      System subfunction PGID_REPORT_STATE -- report the state of this
                              ; driver.  (NOT SUPPORTED)
                              ;
                              ; PARAMETERS
                              ;
                              ;   ON ENTRY:
                              ;      AH = F_SYSTEM
                              ;      AL = SF_REPORT_STATE
                              ;
                              ;   ON EXIT
                              ;      AH = return status (RS_UNSUPPORTED)
                              ;
                              ; REGISTERS ALTERED: ah,dx
                              ;
                              ; DEFINITION MODIFICATION HISTORY
                              ;
                              ;   VERSION:
                              ;
                              ;   DESCRIPTION OF CHANGES:
                              ;
                              ;*****************************************************************
 04BF                         pgid_state         proc     near
 04BF  B4 02                                     mov      ah,RS_UNSUPPORTED        ; function not supported
 04C1  C3                                        ret                        ; return to main driver
 04C2                         pgid_state         endp
                                                 subttl   PGID_VERSION System Subfunction
                                                 page
                              ;***FUNCTION HEADER**********************************************
                              ;
                              ; NAME:          PGID_VERSION
                              ;
                              ; FUNCTIONAL DESCRIPTION
                              ;      System subfunction SF_VERSION_DESC -- Report the version
                              ; number of the driver.  (Use standard system version number)
                              ;
                              ; PARAMETERS
                              ;
                              ;   ON ENTRY
                              ;      AH = F_SYSTEM
                              ;      AL = SF_VERSION_DESC
```

RS-232 Mouse Driver

```
                                    ;    ON EXIT
                                    ;       AH = RS_SUCCESSFUL
                                    ;   (others)  see hp_system_version function.
                                    ;
                                    ;  REGISTERS ALTERED  ah,es,di
                                    ;
                                    ;  DEFINITION MODIFICATION HISTORY
                                    ;
                                    ;    VERSION
                                    ;
                                    ;    DESCRIPTION OF CHANGES
                                    ;
                                    ;**********************************************************************
 04C2                               pgid_version    proc    near
 04C2   B4 00                                       mov     ah, RS_SUCCESSFUL
 04C4   BB 5225                                     mov     bx, 5225H
 04C7   B9 0010                                     mov     cx, VERSION_LEN
 04CA   0E                                          push    cs
 04CB   07                                          pop     es
 04CC   8D 3E 0099 R                                lea     di, cs:VERSION_LAB
 04D0   C3                                          ret                     ; return to PGID main driver
 04D1                               pgid_version    endp


                                    ;**********************************************************************
 04D1                               END_OF_DRIVER:

                                    ;LOCAL STACK USED DURING INITIALIZATION

 04D1       40 [                            DB 64 DUP (0)
                    00
                        ]


 0511                               STACK_TOP:
 0511                               CODE    ENDS

                                            END
```

Structures and records

N a m e	Width Shift	# fields Width	Mask	Initial
ATTR	0010	000B		
DEV	000F	0001	8000	0000
IOCTL	000E	0001	4000	0000
IBM	000D	0001	2000	0000
X	000C	0001	1000	0000
OCREM	000B	0001	0800	0000
Y	0005	0006	07E0	0000
SPEC	0004	0001	0010	0000
CLK	0003	0001	0008	0000
NUL	0002	0001	0004	0000
STDO	0001	0001	0002	0000
STDI	0000	0001	0001	0000
DESCRIBE	0020	0017		
D_SOURCE	0000			
D_HPHIL_ID	0001			
D_DESC_MASK	0002			
D_IO_MASK	0003			
D_XDESC_MASK	0004			
D_MAX_AXIS	0005			
D_CLASS	0006			
D_PROMPTS	0007			
D_RESERVED	0008			
D_BURST_LEN	0009			
D_WR_REG	000A			
D_RD_REG	000B			
D_TRANSITION	000C			
D_STATE	000D			
D_RESOLUTION	000E			
D_SIZE_X	0010			
D_SIZE_Y	0012			
D_ABS_X	0014			
D_ABS_Y	0016			
D_REL_X	0018			
D_REL_Y	001A			
D_ACCUM_X	001C			
D_ACCUM_Y	001E			
HP_ATTR	0010	000D		
HP	000F	0001	8000	0000
DEVCFG	000E	0001	4000	0000
ISR	000D	0001	2000	0000
ENTRY	000C	0001	1000	0000
TYPE	0009	0003	0E00	0000
STR	0008	0001	0100	0000
MAP_CALL	0007	0001	0080	0000
A	0006	0001	0040	0000
SUBADD	0004	0002	0030	0000

RS-232 Mouse Driver

```
   PSHARE . . . . . . . . . . . . .   0003    0001    0008    0000
   CSHARE . . . . . . . . . . . . .   0002    0001    0004    0000
   ROM  . . . . . . . . . . . . . .   0001    0001    0002    0000
   B  . . . . . . . . . . . . . . .   0000    0001    0001    0000
HP_HEADER . . . . . . . . . . . . .   0010    0009
   DH_ATR . . . . . . . . . . . . .   0000
   DH_NAME_INDEX  . . . . . . . . .   0002
   DH_V_DEFAULT . . . . . . . . . .   0004
   DH_P_CLASS . . . . . . . . . . .   0008
   DH_C_CLASS . . . . . . . . . . .   0008
   DH_V_PARENT  . . . . . . . . . .   000A
   DH_V_CHILD . . . . . . . . . . .   000C
   DH_MAJOR . . . . . . . . . . . .   000E
   DH_MINOR . . . . . . . . . . . .   000F
REQ_HEADER  . . . . . . . . . . . .   0017    000A
   RH_LENGTH  . . . . . . . . . . .   0000
   RH_UNIT_CODE . . . . . . . . . .   0001
   RH_CMD_CODE  . . . . . . . . . .   0002
   RH_STATUS  . . . . . . . . . . .   0003
   RH_RESERVED  . . . . . . . . . .   0005
   RH_UNIT_CNT  . . . . . . . . . .   000D
   RH_END_OFF . . . . . . . . . . .   000E
   RH_END_SEG . . . . . . . . . . .   0010
   RH_BPB . . . . . . . . . . . . .   0012
   RH_DRIV  . . . . . . . . . . . .   0016
STATUS  . . . . . . . . . . . . . .   0010    0005
   ERROR  . . . . . . . . . . . . .   000F    0001    8000    0000
   Z  . . . . . . . . . . . . . . .   000A    0005    7C00    0000
   BUSY . . . . . . . . . . . . . .   0009    0001    0200    0000
   DONE . . . . . . . . . . . . . .   0008    0001    0100    0000
   ERR_TYPE . . . . . . . . . . . .   0000    0008    00FF    0000


Segments and Groups

                N a m e               Size    Align   Combine Class

CODE  . . . . . . . . . . . . . . .   0511    PARA    PUBLIC  'CODE'

Symbols

                N a m e               Type    Value   Attr

ABS_MOVE  . . . . . . . . . . . . .   L NEAR  0430    CODE
BAD_CMD . . . . . . . . . . . . . .   L NEAR  0292    CODE
BISR2 . . . . . . . . . . . . . . .   L NEAR  01FA    CODE
BUTTON_DOWN . . . . . . . . . . . .   L NEAR  0487    CODE
BUTTON_ISR  . . . . . . . . . . . .   L NEAR  0470    CODE
BUTTON_TAB  . . . . . . . . . . . .   L BYTE  01F7    CODE
BUTTON_UP . . . . . . . . . . . . .   L NEAR  0481    CODE
CHECK_F_SYSTEM  . . . . . . . . . .   L NEAR  0408    CODE
CMD_TABLE . . . . . . . . . . . . .   L WORD  0152    CODE
COM_MSG . . . . . . . . . . . . . .   L BYTE  00C1    CODE
COM_NUMBER  . . . . . . . . . . . .   L WORD  0128    CODE
CR  . . . . . . . . . . . . . . . .   Number  000D
DEBUG . . . . . . . . . . . . . . .   Alias   TRUE
DEV_ATTR  . . . . . . . . . . . . .   Number  AC18
DEV_DESCRIBE  . . . . . . . . . . .   L 0020  0030    CODE
DEV_DRIVER  . . . . . . . . . . . .   F PROC  0000    CODE    Length =03FF
DEV_HEADER  . . . . . . . . . . . .   L 0010  0020    CODE
DEV_INTERRUPT . . . . . . . . . . .   F PROC  0270    CODE    Length =018F
DEV_STRATEGY  . . . . . . . . . . .   F PROC  0265    CODE    Length =000B
DOS_ENTRY . . . . . . . . . . . . .   Number  C021
DRIVER_ATTR . . . . . . . . . . . .   L WORD  0004    CODE
DRIVER_NAME . . . . . . . . . . . .   L BYTE  000A    CODE
END_OF_DRIVER . . . . . . . . . . .   L NEAR  04D1    CODE
EXIT  . . . . . . . . . . . . . . .   L NEAR  029E    CODE
FALSE . . . . . . . . . . . . . . .   Number  0000
FRAME_COUNT . . . . . . . . . . . .   L WORD  013A    CODE
F_INQUIRE_ENTRY . . . . . . . . . .   Number  000C
F_INS_BASEHPVT  . . . . . . . . . .   Number  0004
F_INS_XCHGFREE  . . . . . . . . . .   Number  000A
F_IO_CONTROL  . . . . . . . . . . .   Number  0004
F_ISR . . . . . . . . . . . . . . .   Number  0000
F_SYSTEM  . . . . . . . . . . . . .   Number  0002
GIVE_TO_PARENT  . . . . . . . . . .   L NEAR  048D    CODE
HPHIL_ADD . . . . . . . . . . . . .   L BYTE  0150    CODE
HPHIL_TABLE . . . . . . . . . . . .   L BYTE  0142    CODE    Length =000E
HP_ENTRY  . . . . . . . . . . . . .   Number  006F
IC_1  . . . . . . . . . . . . . . .   L NEAR  02C8    CODE
IC_10 . . . . . . . . . . . . . . .   L NEAR  0363    CODE
IC_2  . . . . . . . . . . . . . . .   L NEAR  02DA    CODE
IC_3  . . . . . . . . . . . . . . .   L NEAR  02EF    CODE
IC_4  . . . . . . . . . . . . . . .   L NEAR  02F6    CODE
IC_4A . . . . . . . . . . . . . . .   L NEAR  0314    CODE
INIT_3  . . . . . . . . . . . . . .   L NEAR  037C    CODE
INIT_BUTTON_STATE . . . . . . . . .   Number  00FF
INIT_CODE . . . . . . . . . . . . .   L NEAR  02A7    CODE
INIT_EXIT . . . . . . . . . . . . .   L NEAR  03C8    CODE
INIT_NO_PORT  . . . . . . . . . . .   L NEAR  03B2    CODE
INIT_NO_VECTOR  . . . . . . . . . .   L NEAR  03A9    CODE
```

RS-232 Mouse Driver

INIT_OK	L NEAR	03BB	CODE	
INT_ENT	L WORD	0008	CODE	
INT_TABLE	L WORD	012A	CODE	
LAST_SYNCH	L BYTE	0141	CODE	
LF	Number	000A		
MASK_TABLE	L WORD	0132	CODE	
MAX_PGID_SYS_FN	E BYTE	0006		
MBUTTON	L NEAR	01DB	CODE	
MBUTTON_DOWN	L NEAR	01E6	CODE	
MBUTTON_ISR	L NEAR	01E8	CODE	
MBUTTON_UP	L NEAR	01E2	CODE	
MNEXT_BUTTON	L NEAR	021F	CODE	
MOUSE_INT	L NEAR	0172	CODE	
MOVEMENT_ISR	L NEAR	0419	CODE	
MSD_BAD_LENGTH	Number	0005		
MSD_BLD_BPB	Number	0002		
MSD_CRC_ERROR	Number	0004		
MSD_DEV_CLOSE	Number	000E		
MSD_DEV_OPEN	Number	000D		
MSD_GEN_FAILURE	Number	000C		
MSD_INIT	Number	0000		
MSD_INPUT	Number	0004		
MSD_IN_FLUSH	Number	0007		
MSD_IN_NOWAIT	Number	0005		
MSD_IN_STATUS	Number	0006		
MSD_IOCTL_IN	Number	0003		
MSD_IOCTL_OUT	Number	000C		
MSD_MEDIA_CHK	Number	0001		
MSD_NOT_RDY	Number	0002		
MSD_OUTPUT	Number	0008		
MSD_OUT_FLUSH	Number	000B		
MSD_OUT_STATUS	Number	000A		
MSD_OUT_VERIFY	Number	0009		
MSD_PAPER_OUT	Number	0009		
MSD_READ_FAULT	Number	000B		
MSD_REM_MEDIA	Number	000F		
MSD_SEC_NOT_FOUND	Number	0008		
MSD_SEEK_ERROR	Number	0006		
MSD_UNKNOWN_CMD	Number	0003		
MSD_UNKNOWN_MEDIA	Number	0007		
MSD_UNKNOWN_UNIT	Number	0001		
MSD_WRITE_FAULT	Number	000A		
MSD_WRITE_PROT	Number	0000		
MSE_NUM_BUTTON	Number	004C		
MSI_1	L NEAR	01A2	CODE	
MSI_2	L NEAR	01B5	CODE	
MSI_3	L NEAR	0225	CODE	
MSI_4	L NEAR	023D	CODE	
MSI_5	L NEAR	0260	CODE	
NO_PORT_MSG	L BYTE	00C8	CODE	
NO_VECTOR	L BYTE	0103	CODE	
OK_MSG	L BYTE	00AB	CODE	
PGID_DRIVER	N PROC	03FF	CODE	Global Length =0015
PGID_INIT	N PROC	04B4	CODE	Length =0008
PGID_ISR	N PROC	0414	CODE	Length =0082
PGID_OPCODE_BAD	L NEAR	0411	CODE	
PGID_START	N PROC	04BC	CODE	Length =0003
PGID_STATE	N PROC	04BF	CODE	Length =0003
PGID_SYSTEM	N PROC	0496	CODE	Length =001E
PGID_SYS_BAD	L NEAR	04A9	CODE	
PGID_SYS_CASE	L NEAR	04AC	CODE	
PGID_VECT_NUM	L BYTE	0151	CODE	
PGID_VERSION	N PROC	04C2	CODE	Length =000F
PRINT_STR	Number	0009		
REL_MOVE	L NEAR	045C	CODE	
REQ_HDR_OFF	L WORD	0050	CODE	
REQ_HDR_SEG	L WORD	0052	CODE	
RH_CMD_LINE	E DWORD	0012		
RS_DONE	Number	0006		
RS_FAIL	Number	00FE		
RS_NO_VECTOR	Number	00F6		
RS_SUCCESSFUL	Number	0000		
RS_UNSUPPORTED	Number	0002		
SF_MOUSE_OVERRIDE	Number	0002		
SIGN_ON_MSG	L BYTE	0054	CODE	
STACK_PTR	L WORD	0124	CODE	
STACK_SEG	L WORD	0126	CODE	
STACK_TOP	L NEAR	0511	CODE	
STRAT_ENT	L WORD	0006	CODE	
TEMP_BUFFER	L BYTE	013C	CODE	Length =0005
TRUE	Number	- 0001		
T_ABS08	Number	0042		
T_ABS16	Number	0043		
T_KC_BUTTON	Number	0009		
T_REL08	Number	0040		
T_REL16	Number	0041		
UNSUPPORT_CMD	L NEAR	0292	CODE	
UP_DOWN_BIT	Number	0080		
VERSION_LAB	L BYTE	0099	CODE	
VERSION_LEN	Number	0010		
V_DOLITTLE	Number	0006		
V_LHPMOUSE	Number	00CC		

RS-232 Mouse Driver

```
V_SINPUT . . . . . . . . . . . . . .      Number  002A
V_SYSTEM . . . . . . . . . . . . . .      Number  0012

40380 Bytes free

Warning Severe
Errors  Errors
0       0


A . . . . . . . . . . . . . . . . .  203#
ABS_MOVE . . . . . . . . . . . . . .  999    1001    1021#
ATTR . . . . . . . . . . . . . . . .   76#    263

B . . . . . . . . . . . . . . . . .  203#
BAD_CMD . . . . . . . . . . . . . .  613     615     623#
BISR2 . . . . . . . . . . . . . . .  498     502#
BUSY . . . . . . . . . . . . . . . .   92#
BUTTON_DOWN . . . . . . . . . . . . 1093    1097#
BUTTON_ISR . . . . . . . . . . . . .  984    1080#
BUTTON_TAB . . . . . . . . . . . . .  496     499#
BUTTON_UP . . . . . . . . . . . . . 1094#

CHECK_F_SYSTEM . . . . . . . . . . .  907     911#
CLK . . . . . . . . . . . . . . . .   76#
CMD_TABLE . . . . . . . . . . . . .  387#    819
CODE . . . . . . . . . . . . . . . .  253#    253     255     898    1341
COM_MSG . . . . . . . . . . . . . .  338#    702
COM_NUMBER . . . . . . . . . . . . .  360#    432     691     694     759
CR . . . . . . . . . . . . . . . . .  145#    329     332     338     338     348     349     354     873
CSHARE . . . . . . . . . . . . . . .  203#

DEBUG . . . . . . . . . . . . . . .   50#
DESCRIBE . . . . . . . . . . . . . .  166#    197
DEV . . . . . . . . . . . . . . . .   76#
DEVCFG . . . . . . . . . . . . . . .  203#
DEV_ATTR . . . . . . . . . . . . . .  273#    274
DEV_DESCRIBE . . . . . . . . . . . .  284#
DEV_DRIVER . . . . . . . . . . . . .  258#    866
DEV_HEADER . . . . . . . . . . . . .  274#
DEV_INTERRUPT . . . . . . . . . . .  265     600#    865
DEV_STRATEGY . . . . . . . . . . . .  264     592#    596
DH_ATR . . . . . . . . . . . . . . .  153#
DH_C_CLASS . . . . . . . . . . . . .  157#
DH_MAJOR . . . . . . . . . . . . . .  160#
DH_MINOR . . . . . . . . . . . . . .  161#
DH_NAME_INDEX . . . . . . . . . . .  154#
DH_P_CLASS . . . . . . . . . . . . .  156#
DH_V_CHILD . . . . . . . . . . . . .  159#
DH_V_DEFAULT . . . . . . . . . . . .  155#
DH_V_PARENT . . . . . . . . . . . .  158#   1112
DONE . . . . . . . . . . . . . . . .   92#    631
DOS_ENTRY . . . . . . . . . . . . .  140#    661     819     826     835
DRIVER_ATTR . . . . . . . . . . . .  263#
DRIVER_NAME . . . . . . . . . . . .  266#
D_ABS_X . . . . . . . . . . . . . .  190#   1022    1023    1037    1058
D_ABS_Y . . . . . . . . . . . . . .  191#   1029    1030    1038    1059
D_ACCUM_X . . . . . . . . . . . . .  194#
D_ACCUM_Y . . . . . . . . . . . . .  195#
D_BURST_LEN . . . . . . . . . . . .  182#
D_CLASS . . . . . . . . . . . . . .  175#
D_DESC_MASK . . . . . . . . . . . .  171#
D_HPHIL_ID . . . . . . . . . . . . .  170#
D_IO_MASK . . . . . . . . . . . . .  172#
D_MAX_AXIS . . . . . . . . . . . . .  174#
D_PROMPTS . . . . . . . . . . . . .  178#
D_RD_REG . . . . . . . . . . . . . .  184#
D_REL_X . . . . . . . . . . . . . .  192#   1025    1032    1054
D_REL_Y . . . . . . . . . . . . . .  193#   1056
D_RESERVED . . . . . . . . . . . . .  181#
D_RESOLUTION . . . . . . . . . . . .  187#
D_SIZE_X . . . . . . . . . . . . . .  188#
D_SIZE_Y . . . . . . . . . . . . . .  189#   1027
D_SOURCE . . . . . . . . . . . . . .  168#
D_STATE . . . . . . . . . . . . . .  186#   1095    1099    1218
D_TRANSITION . . . . . . . . . . . .  185#   1090
D_WR_REG . . . . . . . . . . . . . .  183#
D_XDESC_MASK . . . . . . . . . . . .  173#

END_OF_DRIVER . . . . . . . . . . .  850    1331#
ENTRY . . . . . . . . . . . . . . .  203#
ERROR . . . . . . . . . . . . . . .   92#    826
ERR_TYPE . . . . . . . . . . . . . .   92#
EXIT . . . . . . . . . . . . . . . .  631#    863

FALSE . . . . . . . . . . . . . . .   48#     49
FRAME_COUNT . . . . . . . . . . . .  371#    439     452     460
F_INQUIRE_ENTRY . . . . . . . . . .  220#    778
F_INS_BASEHPVT . . . . . . . . . . .  212#
F_INS_XCHGFREE . . . . . . . . . . .  213#    793
F_IO_CONTROL . . . . . . . . . . . .  218#    806
F_ISR . . . . . . . . . . . . . . .  216#    521     570     906    1111
F_SYSTEM . . . . . . . . . . . . . .  217#    912
```

RS-232 Mouse Driver

```
GIVE_TO_PARENT . . . . . . . . . .   1039    1064    1098    1110#

HP . . . . . . . . . . . . . . . .   203#
HPHIL_ADD . . . . . . . . . . . .    382#
HPHIL_TABLE . . . . . . . . . . .    378#
HP_ATTR . . . . . . . . . . . . .    203#    273
HP_ENTRY . . . . . . . . . . . . .   225#    524     573     781     798     810     843     1113
HP_HEADER . . . . . . . . . . . .    151#    163

IBM . . . . . . . . . . . . . . .     76#
IC_1 . . . . . . . . . . . . . . .   670#    680
IC_10 . . . . . . . . . . . . . .    772     773#
IC_2 . . . . . . . . . . . . . . .   672     682#
IC_3 . . . . . . . . . . . . . . .   675     678     685     687     694#
IC_4 . . . . . . . . . . . . . . .   692     699#
IC_4A . . . . . . . . . . . . . .    713     718#
INIT_3 . . . . . . . . . . . . . .   784     790#
INIT_BUTTON_STATE . . . . . . . .    1216#   1218
INIT_CODE . . . . . . . . . . . .    387     641#
INIT_EXIT . . . . . . . . . . . .    820     827     837#
INIT_NO_PORT . . . . . . . . . . .   714     822#
INIT_NO_VECTOR . . . . . . . . . .   799     815#
INIT_OK . . . . . . . . . . . . .    813     829#
INT_ENT . . . . . . . . . . . . .    285#
INT_TABLE . . . . . . . . . . . .    382#    760
IOCTL . . . . . . . . . . . . . .     76#
ISR . . . . . . . . . . . . . . .    203#

LAST_SYNCH . . . . . . . . . . . .   376#    482     483
LF . . . . . . . . . . . . . . . .   144#    329     332     338     338     349     349     355     676

MAP_CALL . . . . . . . . . . . . .   203#
MASK_TABLE . . . . . . . . . . . .   386#    770
MAX_PGID_SYS_FN . . . . . . . . .    1159    1100#
MBUTTON . . . . . . . . . . . . .    475#    535
MBUTTON_DOWN . . . . . . . . . . .   480     487#
MBUTTON_ISR . . . . . . . . . . .    485     491#
MBUTTON_UP . . . . . . . . . . . .   482#
MNEXT_BUTTON . . . . . . . . . . .   478     532#
MOUSE_INT . . . . . . . . . . . .    412#    763
MOVEMENT_ISR . . . . . . . . . . .   993#
MSD_BAD_LENGTH . . . . . . . . . .   109#
MSD_BLD_BPB . . . . . . . . . . .    122#
MSD_CRC_ERROR . . . . . . . . . .    108#
MSD_DEV_CLOSE . . . . . . . . . .    134#
MSD_DEV_OPEN . . . . . . . . . . .   133#
MSD_GEN_FAILURE . . . . . . . . .    116#
MSD_INIT . . . . . . . . . . . . .   120#    612
MSD_INPUT . . . . . . . . . . . .    124#
MSD_IN_FLUSH . . . . . . . . . . .   127#
MSD_IN_NOWAIT . . . . . . . . . .    125#
MSD_IN_STATUS . . . . . . . . . .    126#
MSD_IOCTL_IN . . . . . . . . . . .   123#
MSD_IOCTL_OUT . . . . . . . . . .    132#
MSD_MEDIA_CHK . . . . . . . . . .    121#
MSD_NOT_RDY . . . . . . . . . . .    106#
MSD_OUTPUT . . . . . . . . . . . .   128#
MSD_OUT_FLUSH . . . . . . . . . .    131#
MSD_OUT_STATUS . . . . . . . . . .   130#
MSD_OUT_VERIFY . . . . . . . . . .   129#
MSD_PAPER_OUT . . . . . . . . . .    113#
MSD_READ_FAULT . . . . . . . . . .   115#
MSD_REM_MEDIA . . . . . . . . . .    135#    614
MSD_SEC_NOT_FOUND . . . . . . . .    112#
MSD_SEEK_ERROR . . . . . . . . . .   110#
MSD_UNKNOWN_CMD . . . . . . . . .    107#    627
MSD_UNKNOWN_MEDIA . . . . . . . .    111#
MSD_UNKNOWN_UNIT . . . . . . . . .   105#
MSD_WRITE_FAULT . . . . . . . . .    114#
MSD_WRITE_PROT . . . . . . . . . .   104#
MSE_NUM_BUTTON . . . . . . . . . .   199#    845
MSI_1 . . . . . . . . . . . . . .    441     446     450#
MSI_2 . . . . . . . . . . . . . .    454     459#
MSI_3 . . . . . . . . . . . . . .    465     541#
MSI_4 . . . . . . . . . . . . . .    550#
MSI_5 . . . . . . . . . . . . . .    440     455     551     570#

NO_PORT_MSG . . . . . . . . . . .    340#    824
NO_VECTOR . . . . . . . . . . . .    350#    817
NUL . . . . . . . . . . . . . . .     76#

OCREM . . . . . . . . . . . . . .     76#
OK_MSG . . . . . . . . . . . . . .   334#    833

PGID_DRIVER . . . . . . . . . . .    787     899     905#    926
PGID_INIT . . . . . . . . . . . .    1176    1215#   1222
PGID_ISR . . . . . . . . . . . . .   908     979#    1115
PGID_OPCODE_BAD . . . . . . . . .    913     922#
PGID_START . . . . . . . . . . . .   1177    1252#   1256
PGID_STATE . . . . . . . . . . . .   1178    1286#   1289
PGID_SYSTEM . . . . . . . . . . .    914     1157#   1181
PGID_SYS_BAD . . . . . . . . . . .   1160    1168#
PGID_SYS_CASE . . . . . . . . . .    1167    1175#   1180
PGID_VECT_NUM . . . . . . . . . .    383#    515     566     804
PGID_VERSION . . . . . . . . . . .   1179    1319#   1327
```

RS-232 Mouse Driver

```
PRINT_STR                 139#   660    818    825    834
PSHARE                    203#

REL_MOVE                  995    997    1053#
REQ_HDR_OFF               315#   593    610    849
REQ_HDR_SEG               316#   594
REQ_HEADER                 56#    69
RH_BPB                     66#    71
RH_CMD_CODE                60#   611
RH_CMD_LINE                71#   667
RH_DRIV                    67#
RH_END_OFF                 64#   850
RH_END_SEG                 65#   851
RH_LENGTH                  58#
RH_RESERVED                62#
RH_STATUS                  61#   626    627    631
RH_UNIT_CNT                63#
RH_UNIT_CODE               59#
ROM                       203#
RS_DONE                   239#
RS_FAIL                   240#   1006
RS_NO_VECTOR              241#   798
RS_SUCCESSFUL             237#   1220   1254   1320
RS_UNSUPPORTED            238#   783    923    1169   1287

SF_MOUSE_OVERRIDE         223#   807
SIGN_ON_MSG               318#   659
SPEC                       76#
STACK_PTR                 357#   646    856
STACK_SEG                 358#
STACK_TOP                 651    1340#
STATUS                     92#
STDI                       76#
STDO                       76#
STR                       203#
STRAT_ENT                 264#
SUBADD                    203#

TEMP_BUFFER               372#   450    461    542    544    546    548
TRUE                       49#    50
TYPE                      203#

T_ABS08                   232#   998
T_ABS16                   233#   1000
T_KC_BUTTON               229#   514    983
T_REL08                   230#   994
T_REL16                   231#   564    996

UNSUPPORT_CMD             388    389    390    391    392    393    394    395    396    397    398    399    400    401
                          402    624#

UP_DOWN_BIT               1078#  1092

VERSION_LAB               330#   333    1325
VERSION_LEN               333#   1322
V_DOLITTLE                209#   274
V_LHPMOUSE                222#   274    808    844
V_SINPUT                  215#   522    571    779
V_SYSTEM                  211#   794

X                          76#

Y                          76#

Z                          92#

220 Symbols

50960 Bytes Free
```

Glossary

ACK - Acknowledge.

Adapter - A circuit board containing electronic circuitry that interfaces a peripheral to the system processor board.

Adapter Card - See ADAPTER

Alphanumeric Display Mode - One of the Video Display Adapter modes. When this mode is selected, data is displayed in character cells, organized in rows and columns on the screen.

Application Programs - Software that performs application-specific tasks. Word processors, spreadsheets, and data bases are examples of application programs.

Barcode Reader - An input device that is used to scan surfaces containing barcodes. The barcode reader converts barcodes into scancode data format, and transmits the scancodes to an input interface.

Baud Rate - The rate a signal changes state. When used with relationship to RS-232 ports, it is synonymous with the data transfer rate, expressed in bits per second (BPS).

BCD - Binary Coded Decimal.

BIOS - Basic Input/Output System. The BIOS is the code module that contains the drivers that constitute the software interface between the hardware, and system software and application programs.

Bootstrap - The process of initializing the system and loading system software after a reset.

BPS - Bits per second.

Bucket - A data structure used by the EX-BIOS string functions for alphanumeric string management.

CALL SYSCALL - Issues an HP system call. This routine assumes that the EX-BIOS is enabled. When first called, this routine will patch the first instruction "JMP SHORT PATCH" to become "INT XXH" where XXH is the current HP interrupt number.

Character Code - A word returned by the keyboard driver indicating a key stroke. The character code consists of a keyboard scancode, and either an Extended (00H) or ASCII character.

Checksum - An error-checking protocol used to verify the integrity of a block of data or code. Each byte or word in the block is summed, then added to a checksum byte. The block of data or code is presumed valid if this sum equals a predefined value, usually 0.

Checksum Byte - A byte added to the sum of a block of code or data to produce a valid sum.

Child Driver - A child driver is called by another driver when that driver is unable to perform a function requested of it. Child drivers perform lower level or more hardware-specific tasks than their calling drivers.

Clipping - The process utilized when dealing with graphics coordinates outside of the logical coordinate space. The Input System clips coordinates so that they don't exceed the boundaries of the logical coordinate space.

CMOS Memory - RAM memory on the Processor Board that is powered by both the system power supply and battery. When the system power is turned off, the contents of the RAM memory are preserved by the battery.

Code Module - A group of related processor instructions.

Code Segment (CS) - The segment address of the code module currently being executed.

Compatibility Function Keys - The 10 function keys labeled F1-F10 on the keyboard. The HP Function keys (f1-f8) can be mapped to emulate their respective Compatibility Function keys (F1-F8). See also HP FUNCTION KEYS.

Coprocessor - An add-on processor that works with the CPU (Central Processing Unit) found on the Processor PCA. The 80287 (Vectra ES), the 80387 (Vectra QS and RS), and the Weitek coprocessor (Vectra RS only) are examples of specialized coprocessors for floating point arithmetic.

CRC - See CYCLIC REDUNDANCY CHECK CHARACTER.

CS - See CODE SEGMENT.

Cursor Control Keypad - The keypad containing the HP cursor control keys.

Cylinder - A term used with multi-platter disc mechanisms, a cylinder is a group of sectors having the same track number on each of the platters.

Cyclic Redundancy Check Character - Character used as a redundant character for error detection in various modified cyclic codes.

Daisy Chain - A method of linking devices together in a serial configuration. Input devices on the HP-HIL loop are connected in a daisy chain.

DASD - Direct Access Standard Device.

Data Segment (DS) - The segment address of the data currently being accessed.

Data Structures - A related group of data fields.

Describe Record - A data structure utilized by the Input System which contains information characterizing an input event.

Device - A physical piece of hardware, e.g., a Touchscreen, mouse, keyboard, dot matrix printer, ThinkJet, or LaserJet.

Disc Partitions - A group of cylinders within a hard disc volume allocated to a specified operating system, and its associated programs and data.

Disc Volumes - A group of cylinders comprising a logical disc. The optional 20 Mbyte hard disc contains a single volume. Optional hard discs greater than 32 Mbytes in size must be divided into two or more volumes of up to 32 Mbytes each.

Divide By Zero Interrupt - The CPU executes this interrupt any time a divide-by-zero operation is attempted. The vector to the service routine for this interrupt must be stored in memory locations 0000:0000H-0000:0003H.

DMA - Direct Memory Access.

DOS - Disc Operating System.

DOS Installable Device Driver - A device driver designed to be dynamically installed by DOS. DOS installable device drivers may be used to add EX-BIOS drivers to the system.

Driver - Code that interfaces to either a physical device or another driver.

Driver Header - A data structure contained in the data area of each EX-BIOS driver. The driver header contains data fields that specify the attributes, mapping, and other parameters of the driver.

DS - Driver Segment.

EGA - Enhanced Graphics Adapter.

EOI - End Of Interrupt.

EOT - End Of Track.

EX-BIOS - Extended BIOS. A set of HP proprietary drivers that provide support for various system features.

Extra Segment (ES) - The segment address of the extra data segment currently being accessed.

FAT - File Allocation Table.

FDC - Flexible Disc Controller.

Functions - Code modules within a driver that perform specific tasks. Individual driver functions are selected when a driver is called.

Function Keys - The keys (F1-F12) on the Vectra Enhanced keyboard. See also HP FUNCTION KEYS, and COMPATIBILITY FUNCTION KEYS.

GDT - Global Descriptor Table.

GID - see GRAPHIC INPUT DEVICE.

Graphic Display Mode - A video display adapter mode in which all positions on the screen are addressable as pixels.

Graphic Input Device - An input device that generates positional and/or button state data. A mouse, tablet, and touchscreen are examples of graphic input devices.

Graphics Sprite - See SPRITE.

Hardware Interrupts - Requests for interrupt service generated by the hardware components.

Head - The magnetic device that reads and writes data from a disc drive. Disc drives have a head for each recording surface in the mechanism. A flexible disc has two heads, while a hard disc head count can vary depending on the drive being used. The optional 20MB disc has two platters and four heads.

Hexadecimal - Numbers expressed in base 16. Hexadecimal notation is used throughout this manual to represent binary data. Hexadecimal digits are represented with the numbers 0-9 and letters A-F. The hexadecimal numbers are indicated with an uppercase 'H' as their last character (i.e., 17H).

HP Extensions - Additional functions added to industry standard drivers that support EX-BIOS features and/or provide additional flexibility in programming industry standard system capabilities.

HP Function Keys - The function keys labeled f1-f8 on the **Vectra Keyboard/DIN**. These keys can be mapped to return their own scancode, or they may emulate their respective Compatibility Function keys (F1-F8). See also COMPATIBILITY FUNCTION KEYS.

HP Global Data Area - A data structure located in the EX-BIOS Data Area containing variables common to two or more EX-BIOS drivers. In addition, the stack used by the EX-BIOS drivers is located here.

HP_ENTRY_CODE - The code module that dispatches the EX-BIOS interrupt (default 6FH) to the selected driver.

HP_ENTRY - The symbolic reference for the EX-BIOS interrupt (default 6FH). Always use a "CALL SYSCALL" routine to call the EX-BIOS drivers.

HP-HIL Controller - The hardware that provides the electrical interface to the HP-HIL link and supervises the communication protocol.

HP-HIL Link - The electrical interface and communication protocol utilized to connect HP-HIL input devices.

HP-HIL Major Address - The primary address of an HP-HIL device. This is typically the link address of the device.

HP-HIL Minor Address - The secondary address of an HP-HIL device.

HP-HIL Universal Address - Used to broadcast commands to all HP-HIL devices. The Universal Address is implemented as Address 0 in the HP-HIL protocol.

HP_VECTOR_TABLE - A data structure containing the IP, CS, and DS of all EX-BIOS drivers. This data structure is utilized by the HP_ENTRY_CODE to branch to the selected EX-BIOS driver.

Input System - A set of EX-BIOS drivers that service the input devices. The Input System supports the HP Mouse, HP Touchscreen, HP Tablet and other HP-HIL input devices. It can be expanded to encompass non-HP-HIL input devices.

Instruction Pointer - (IP) The offset from the base of the code segment of the next instruction to be executed.

Interleave - The number of physical sectors on a disc drive skipped when reading consecutive logical sectors on the same track. See also STAGGER.

Interrupt Service Routine - A code module, and its associated data structure(s) that responds to a hardware interrupt.

Interrupt Vector - A data structure used by the CPU to branch to a service routine or an interrupt. Interrupt vectors are located in the first 1024 bytes of system memory. Each interrupt vector occupies 2 words of memory and contains the IP and CS of the interrupt service routine.

IP - Instruction Pointer.

IRET - Interrupt Return.

IRQ - Interrupt Request.

IS - Industry Standard. Also see **INDUSTRY STANDARD.**

ISR Event Record - A data structure used by the Input System which contains information characterizing an input event.

KB - Kilobytes. 1024 bytes.

Keyboard - The physical keyboard.

Keyboard Controller (8042) - The 8042 keyboard controller. The 8042 provides industry standard keyboard compatibility and serves as a buffer between the STD-BIOS keyboard drivers and the Input System.

Keyboard Modifier - One of the special keyboard keys that modifies the interpretation of the other keys. The keyboard modifiers are the <CTRL>, <Alt>, <Shift>, <Caps lock>, <Num lock>, and <Scroll Lock> keys.

LED Mode Indicators - The LEDs located on the keyboard that indicate the state of the CAPS LOCK, NUM LOCK, and SCROLL LOCK keyboard modifiers.

Logical Driver - A driver responsible for interfacing with the Operating System or application.

Logical Keyboard - A set of drivers within the Input System that service the physical keyboard.

MB - MegaByte. 1,048,576 bytes.

MASM - Microsoft Macro Assember.

MICKIES - The number of physical coordinates per inch reported by a mouse or other relative graphics input device (GID).

Mouse - A graphics input device (GID) device that reports relative motion coordinates based on its motion. A mouse will also report the state of its buttons.

MS-DOS - Microsoft Disc Operating System. See DOS.

Multi-Tasking - The ability of a CPU to perform multiple jobs or tasks simultaneously. Multi-tasking is accomplished by dividing CPU execution time between the different tasks. If this task-switching is performed quickly enough, the illusion of simultaneous execution occurs.

Numeric Keypad - The keypad containing numeric and modifier keys.

NMI - Non-Maskable Interrupt. This is a CPU interrupt line used to report system error conditions. This interrupt is mapped by the CPU to Interrupt vector 02H.

NOP - No operation. A no-operation instruction causing the computer to do nothing except go to the next instruction.

OBF - Output buffer full.

Operating System - The system software that provides access to system resources for application programs. The operating system manages input and output, data and program files, and system memory.

Original Vectra PC - The precursor to the Vectra ES, QS, and RS series of computers. The original Vectra PC simply had "Vectra" in red letters on its nameplate.

Palette - The set of all possible colors the Video Display Adapter can produce. The Multimode Video Display Adapter has a palette of 16 colors.

Parallel Port - An I/O port that transmits and receives data a byte at a time. The parallel ports are typically used to interface to printers.

Parent Driver - A parent driver is called by another driver when the second is unable to perform a function requested of it. Parent drivers perform higher level or more system software oriented tasks than their calling drivers.

Physical Driver - A driver responsible for interfacing with the physical hardware.

Pixel - A dot on the screen in the graphics modes.

Polling - The process of periodically determining the status of a device. Polling is used to determine if peripheral devices have data or are ready to accept data in non-interrupt driven systems.

POST - Power-On Self Test. The POST process is executed each time the system is powered on.

Processor Interrupts - Interrupts generated by the CPU processor in response to error conditions or processor exceptions.

Protected Mode - One of the two modes that the CPU can operate in. The Protected mode provides virtual memory addressing, in-chip memory management and protection, and task switching to support multi-user, multi-tasking system software.

RAM BIOS - The interface between DOS and the ROM BIOS. It is dynamically loaded at system boot with DOS.

Real Mode - One of the two modes that the CPU can operate in. The Real mode provides compatibility with the 8086 family of microprocessors.

Real-Time Clock - A clock circuit that maintains the correct time whether the system is on or off. The real-time clock is powered by both the system power supply and battery. When the system power is turned off, the clock continues to operate from the battery.

Return Status Code - A code returned by the EX-BIOS drivers that indicates the status of the function requested.

ROM BIOS - The set of EX-BIOS and STD-BIOS drivers. These code modules are contained in the base ROM modules on the Processor PCA.

ROM Module - Code and/or data stored in an EPROM or ROM.

RS-232C - An Electronic Industries Association (EIA) standard for a serial data transmission interface. Often used as a synonym for serial when referring to system ports.

RTC - Real-Time Clock.

Scaling - The process of adjusting physical graphics coordinates to fit in a proportionately larger or smaller logical space. The Input System scales the coordinates received from a tablet to fit into its logical space.

Scancodes - Codes returned by the physical keyboard to indicate key makes and breaks.

SDLC - Synchronous Data Link Control.

Sector - A physical location on the disc where a block of data is stored. Disc surfaces are divided into concentric rings called tracks. These rings are in turn divided into sectors.

Serial - To transmit data one bit at a time, serially. Used to indicate system ports that transmit data in this fashion. See also RS-232C.

Single Step Interrupt - A processor interrupt generated after each instruction if the Single Step flag is set. This interrupt is mapped by the CPU to Interrupt vector 01H.

Software Interrupts - Interrupts generated by the CPU INT 'n' instruction where 'n' is the interrupt number.

Sprite - A graphics cursor. The sprite is controlled by the Input System V_STRACK driver.

SPU - System Processing Unit.

Stagger - Disc stagger is the track to track offset between logical sectors. Stagger increases disc performance during sequential read operations by adjusting for track to track access time. See also INTERLEAVE.

STD-BIOS - The set of drivers that execute the industry standard BIOS functions.

SYSGEN - System generation process.

System Software - See Operating System.

System Strings - Character strings stored in memory. Each EX-BIOS driver has a system string associated with it. System strings are designed to provide a simple method for system software to access them. In addition, their implementation provides a simple and effective method of localization.

Tablet - A Graphics Input Device (GID) that generates absolute graphics coordinates.

Timeout - An indication (for example, an interrupt) that indicates that a predetermined time has elapsed waiting for an event to occur. Timeouts are used to prevent the system from hanging up waiting for an event to happen that doesn't. For example, a timeout can be used to abort a print operation if the printer does not return a ready status.

Timer Tick - An interrupt generated by the system timer. It is initialized to produce approximately 18.2 timer ticks per second.

Touch Screen - An HP Graphic Input Device (GID). Allows a user to input data by physically touching the display screen.

Track - An Input System driver that moves a Sprite on the display screen in response to graphics motion received from GID devices.

Tracking - The process of moving a Sprite on the display screen in response to graphic motion received from GID devices.

Typematic Delay - The amount of time a key must remain depressed before the keyboard enters the typematic or repeat mode.

Typematic Rate - The rate at which make scancodes are transmitted by the keyboard when it is in the typematic or repeat mode.

Video Attributes - Video characteristics of characters displayed on the Video Display Adapter. Video attributes include reverse video, blinking, underline, and high intensity. Video attributes only apply to characters displayed in the alphanumeric modes.

References

HP Vectra MS-DOS User's Reference Manual

HP Vectra MS-DOS Programmer's Reference Manual
--Discusses programming of the CPU using MS-DOS.

HP-HIL Technical Reference Manual
--Discusses the HP-HIL controller.
--Discusses the HP-HIL link.

HP Vectra MS-DOS Macro Assembler
--Reference for the assembler.

INTEL iAPX286 Programmer's Reference Manual
--Reference for CPU instruction set and architecture.
--Reference for the 80287 numeric processor.

INTEL 80386 Programmer's Reference Manual
--Reference for 80386 instruction set and architecture
--Reference for the 80387 numeric processor.

INTEL iAPX286 Hardware Reference Manual
--Discusses the 80286 processor.

INTEL 80386 Hardware Reference Manual
--Discusses the 80386 processor.

INTEL Microsystem Components Handbook, Volume II
--Discusses the 8254 timer chip.
--Discusses the 8042 keyboard controller chip.

INTEL Microprocessor and Peripheral Handbook, Volume I
--Discusses the 8237A DMA controller.
--Discusses the 82284 clock chip.

INTEL the 8086 Family User's Manual

Motorola Single Chip Microcomputer Data, Section C
--Discusses the MC146818 real time clock/ CMOS chip.

Motorola 8-Bit Microprocessor & Peripheral Data
--Discusses the 6845A video controller chip.

NEC Electronics Microcomputer Products Data Book
--Discusses the 765A flexible disc controller chip.

The Peter Norton Guide to the IBM PC by Peter Norton, Microsoft Press.

Writing MS-DOS Device Drivers by Robert S. Lai, Addison-Wesley Publishing Co.

Index

HEWLETT
PACKARD

Printed in USA 01/88
Part Number 45945-90012