Section 9. Compatibility

Notes:

Introduction

This section discusses the major differences between the systems in the IBM Personal Computer and Personal System/2 product lines. Also included are programming considerations that must be taken into account when designing application programs for the IBM Personal Computer and Personal System/2 products.

System Board

The Model 50 and Model 60 system boards use an 80286 microprocessor and have provisions to install an optional 80287 math coprocessor. These 80286/80287 systems are generally compatible with applications written for the 8087 Math Coprocessor. Math coprocessor compatibility is discussed in "8087 to 80287 Compatibility" on page 3-7.

The 80286 system microprocessor and general architecture of the Model 50 and Model 60 have created some fundamental differences between them and other systems. These differences must be taken into consideration when designing programs exclusively for these Personal System/2 products or programs compatible across the IBM Personal Computer and Personal System/2 product lines. Programming considerations are discussed in "Application Guidelines" on page 9-6.

Diskette Drives and Controller

The following figure shows the read, write, and format capabilities for each type of diskette drive used by the Model 50 and Model 60.

Diskette Drive Type	160/180K Mode	320/360K Mode	1.44M Mode	720K Mode
5.25-Inch Diskette Drive:				
Single Sided (48 TPI)	R W F	---	---	---
Double Sided (48 TPI)	R W F	R W F	---	---
3.5-Inch Diskette Drive:				
720KB Drive	---	---	---	R W F
1.44MB Drive	---	---	R W F	R W F

R-Read W-Write F-Format

Figure 9-1. Diskette Drive Read, Write, and Format Capabilities

Notes:

1. 5.25-inch diskettes designed for the 1.2M mode cannot be used in either a 160/180K or a 320/360K diskette drive.
2. 3.5-inch diskettes designed for the 1.44M mode cannot be used in a 720K diskette drive.

Warning: 16-bit operations to the video subsystem can cause a diskette overrun in the 1.44M mode because data width conversions may require more than 12 microseconds. If an overrun occurs, BIOS returns an error code and the operation should be retried.

Copy Protection

The following methods of copy protection may not work on systems using the 5.25-inch high capacity diskette drive or the 3.5-inch 1.44M diskette drive.

- Bypassing BIOS Routines
  - Track Density: The 5.25 inch high capacity diskette drive records tracks at a density of 96 tracks per inch (TPI). This drive has to double-step in the 48 TPI mode, which is performed by BIOS.

- Data Transfer Rate: BIOS selects the proper data transfer rate for the media being used.
- Diskette Parameters Table: Copy protection, which creates its own Diskette Parameters Table may not work on these drives.

- Diskette Drive Controls
  - Rotational Speed: The time between two events on a diskette is a function of the controller.
  - Access Time: Diskette BIOS routines must set the track-to-track access time for the different types of media used in the drives.
  - Diskette Change Signal: Copy protection may not be able to reset this signal.
- Write Current Control—Copy protection that uses write current control will not work because the controller selects the proper write current for the media being used.

Fixed Disk Drives and Controller

Reading from and writing to the fixed disk drive is initiated in the same way as with IBM Personal Computer products; however, new functions are supported. Detailed information about specific fixed disk drives and fixed disk adapters is available in separate technical references.

Application Guidelines

Use the following information to develop application programs for the IBM Personal Computer and Personal System/2 products. Whenever possible, BIOS should be used as an interface to hardware in order to provide maximum compatibility and portability of applications across systems.

Hardware Interrupts

Hardware interrupts are level-sensitive for systems using the Micro Channel architecture while systems using the Personal Computer type I/O channel design have edge-sensitive hardware interrupts. On edge-sensitive interrupt systems, the interrupt controller clears its internal interrupt-in-progress latch when the interrupt routine sends an End-of-Interrupt (EOI) command to the controller. The EOI is sent whether the incoming interrupt request to the controller is active or inactive.

In level-sensitive systems, the interrupt-in-progress latch is readable at an I/O address bit position. This latch is read during the interrupt service routine and may be reset by the read operation or may require an explicit reset.

Note: Designers may want to limit the number of devices sharing an interrupt level for performance and latency considerations.

The interrupt controller on level-sensitive systems requires the interrupt request to be inactive at the time the EOI is sent; otherwise, a "new" interrupt request will be detected and another microprocessor interrupt caused.

To avoid this problem, a level-sensitive interrupt handler must clear the interrupt condition (usually by a Read or Write to an I/O port on the device causing the interrupt). After clearing the interrupt condition, a JMP $+2 should be executed prior to sending the EOI to the interrupt controller. This ensures that the interrupt request is removed prior to re-enabling the interrupt controller. Another JMP $+2 should be executed after sending the EOI, but prior to enabling the interrupt through the set Set Interrupt Enable Flag (STI) command.

In the level-sensitive systems, hardware prevents the interrupt controllers from being set to the edge-sensitive mode.

Hardware interrupt IRQ9 is defined as the replacement interrupt level for the cascade level IRQ2. Program interrupt sharing should be implemented on IRQ2, interrupt hex 0A. The following processing occurs to maintain compatibility with the IRQ2 used by IBM Personal Computer products:

1. A device drives the interrupt request active on IRQ2 of the channel.
2. This interrupt request is mapped in hardware to IRQ9 input on the second interrupt controller.
3. When the interrupt occurs, the system microprocessor passes control to the IRQ9 (interrupt hex 71) interrupt handler.
4. This interrupt handler performs an end of interrupt (EOI) to the second interrupt controller and passes control to IRQ2 (interrupt hex 0A) interrupt handler.
5. This IRQ2 interrupt handler, when handling the interrupt, causes the device to reset the interrupt request prior to performing an EOI to the master interrupt controller that finishes servicing the IRQ2 request.

Software Interrupts

With the advent of software interrupt sharing, software interrupt routines must daisy chain interrupts. Each routine must check the function value and if it is not in the range of function calls for that routine, it must transfer control to the next routine in the chain. Because software interrupts are initially pointed to 0:0, before daisy chaining, it is necessary to check for this case. If the next routine is pointed to 0:0 and the function call is out of range, the appropriate action is to set the carry flag and do a RET 2 to indicate an error condition.

High-Level Language Considerations

The IBM-supported languages of IBM C, BASIC, FORTRAN, COBOL, and Pascal are the best choices for writing compatible programs.

If a program uses specific features of the hardware, that program may not be compatible with all IBM Personal Computer and Personal System/2 products. Specifically, the use of assembler language subroutines or hardware-specific commands (In, Out, Peek, Poke, ...) must follow the assembler language rules. See "Assembler Language Programming Considerations" on page 9-8.

Any program that requires precise timing information should obtain it through an operating system or language interface; for example, TIME$ in BASIC. If greater precision is required, the assembler techniques in "Assembler Language Programming Considerations" are available. The use of programming loops may prevent a program from being compatible with other IBM Personal Computer products, IBM Personal System/2 products, and software.

Assembler Language Programming Considerations

This section describes fundamental differences between the systems in the Personal Computer and Personal System/2 product lines that may affect program development.

Opcodes

The following opcodes work differently on systems using the 80286 microprocessor than they do on systems using the 8088 or 8086 microprocessor.

- PUSH SP

  The 80286 microprocessor pushes the current stack pointer; the 8088 and 8086 microprocessors push the new stack pointer, that is, the value of the stack pointer after the PUSH SP instruction is completed.

- Single step interrupt (when TF = 1) on the interrupt instruction (Opcode hex CC, CD):

The 80286 microprocessor does not perform a single-step interrupt on the INT instruction; the 8088 and 8086 microprocessors do perform a single-step interrupt on the INT instruction.

- The divide error exception (interrupt 0):

  The 80286 microprocessor pushes the CS:IP of the instruction that caused the exception; the 8088 and 8086 microprocessors push the CS:IP of the instruction following the instruction that caused the exception.

- Shift counts for the 80286 microprocessor:

  Shift counts are masked to 5 bits. Shift counts greater than 31 are treated mod 32. For example, a shift count of 36 shifts the operand four places.

- Multiple lockout instructions:

  There are several microprocessor instructions that, when executed, lock out external bus signals. DMA requests are not honored during the execution of these instructions. Consecutive instructions of this type prevent DMA activity from the start of the first instruction to the end of the last instruction. To allow for necessary DMA cycles, as required by the diskette controller in a multitasking system, multiple lock-out instructions must be separated by a JMP SHORT $+2.

- Back-to-back I/O commands:

  Back-to-back I/O commands to the same I/O ports do not permit enough recovery time for some I/O adapters. To ensure enough time, a JMP SHORT $+2 must be inserted between IN/OUT instructions to the same I/O adapters.

  Note: MOV AL,AH type instruction does not allow enough recovery time. An example of the correct procedure follows:

```
OUT  IO_ADD,AL
JMP  SHORT $+2
MOV  AL,AH
OUT  IO_ADD,AL
```

- I/O commands followed by an STI instruction:

  I/O commands followed immediately by an STI instruction do not permit enough recovery time for some system board and channel operations. To ensure enough time, a JMP SHORT $+2 must be inserted between the I/O command and the STI instruction.

  Note: MOV AL,AH type instruction does not allow enough recovery time. An example of the correct procedure follows:

```
OUT  IO_ADD,AL
JMP  SHORT $+2
MOV  AL,AH
STI
```

80286 Anomalies

In the Protected Mode, when any of the null selector values (0000H, 0001H, 0002H, 0003H) are loaded into the DS or ES registers with a MOV or POP instruction or a task switch, the 80286 always loads the null selector 0000H into the corresponding register.

If a coprocessor (80287) operand is read from an "executable and readable" and conforming (ERC) code segment, and the coprocessor operand is sufficiently near the segment limit that the second or subsequent byte lies outside the limit, no protection exception #9 will be generated.

The following describes the operation of all 80286 parts:

- Instructions longer than 10 bytes (instructions using multiple redundant prefixes) generate exception #13 (General Purpose Exception) in both the Real Address Mode and Protected Mode.
- If the second operand of an ARPL instruction is a null selector, the instruction generates an exception #13.

ROM BIOS and Operating System Function Calls

For maximum portability, programs should perform all I/O operations through operating system function calls. In environments where the operating system does not provide the necessary programming interfaces, programs should access the hardware through ROM BIOS function calls, if permissible.

- In some environments, program interrupts are used for access to these functions. This practice removes the absolute addressing from the program. Only the interrupt number is required.
- The coprocessor detects six different exception conditions that can occur during instruction execution. If the appropriate exception mask within the coprocessor is not set, the coprocessor sets the 'error' signal. This 'error' signal generates a hardware interrupt 13 (IRQ 13) causing the 'busy' signal to be held in the busy state. The 'busy' signal can be cleared by an 8-bit I/O Write command to address hex 00F0 with bits D0 through D7 equal to 0.

  The power-on self-test code in the system ROM enables hardware IRQ 13 and sets up its vector to point to a routine in ROM. The ROM routine clears the 'busy' signal latch and then transfers control to the address pointed to by the NMI vector. This maintains code compatibility across the IBM Personal Computer and Personal System/2 product lines. The NMI handler reads the status of the coprocessor to determine if the NMI was caused by the coprocessor. If the interrupt was not caused by the coprocessor, control is passed to the original NMI handler.
- In systems using the 80286 microprocessor, IRQ 9 is redirected to INT hex 0A (hardware IRQ 2). This ensures that hardware designed to use IRQ 2 will operate in these systems. See "Hardware Interrupts" on page 9-6 for more information.
- The system can mask hardware sensitivity. New devices can change the ROM BIOS to accept the same programming interface on the new device.
- In cases where BIOS provides parameter tables, such as for video or diskette, a program can substitute new parameter values by building a new copy of the table and changing the vector to point to that table. However, the program should copy the current table, using the current vector, and then modify those locations in

the table that need to be changed. In this way, the program does not inadvertently change any values that should be left the same.

- The Diskette Parameters Table pointed to by INT hex 1E consists of 11 parameters required for diskette operation. It is recommended that the values supplied in ROM be used. If it becomes necessary to modify any of the parameters, build another parameter block and modify the address at INT hex 1E (0:78) to point to the new block.

  The parameters were established to allow:

  - Some models of the IBM Personal Computer to operate both the 5.25-inch high capacity diskette drive (96 tracks per inch) and the 5.25-inch double-sided diskette drive (48 tracks per inch).
  - Some models of the Personal System/2 to operate both the 3.5-inch 1.44M diskette drive and the 3.5-inch 720KB diskette drive.

  The Gap Length Parameter is not always retrieved from the parameter block. The gap length used during diskette read, write, and verify operations is derived from within diskette BIOS. The gap length for format operations is still obtained from the parameter block.

  Note: Special considerations are required for format operations. Refer to the diskette section of the *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference* for the required details.

  If a parameter block contains a head settle time parameter value of 0 milliseconds, and a write or format operation is being performed, the following minimum head settle times are enforced.

Drive Type	Head Settle Time
5.25-Inch Diskette Drives:	
Double Sided (48 TPI)	20 milliseconds
High Capacity (96 TPI)	15 milliseconds
3.5-Inch Diskette Drives:	
720K	20 milliseconds
1.44M	15 milliseconds

Figure 9-2. Write and Format Head Settle Time

Read and verify operations use the head settle time provided by the parameter block.

If a parameter block contains a motor start wait parameter of less than 500 milliseconds (1 second for a Personal Computer product) for a write or verify operation, diskette BIOS enforces a minimum time of 500 milliseconds (1 second for a Personal Computer product). Read and write operations use the motor start time provided by the parameter block.

- Programs may be designed to reside on both 5.25-inch or 3.5-inch diskettes. Since not all programs are operating-system dependent, the following procedure can be used to determine the type of media inserted into a diskette drive.

  1. Verify Track 0, Head 0, Sector 1 (1 sector): This allows diskette BIOS to determine if the format of the media is a recognizable type.

     If the verify operation fails, issue the reset function (AH = 0) to diskette BIOS and try the operation again. If another failure occurs, the media needs to be formatted or is defective.

  2. Verify Track 0, Head 0, Sector 16 (1 sector).

     If the verify operation fails, either a 5.25-inch (48 TPI) or 3.5-inch 720KB diskette is installed. The type can be determined by verifying Track 78, Head 1, Sector 1 (1 sector). A successful verification of Track 78 indicates a 3.5-inch 720KB diskette is installed; a verification failure indicates a 5.25-inch (48 TPI) diskette is installed.

     Note: Refer to the *DOS Technical Reference* for the File Allocation Table parameters for single-sided and double-sided diskettes.

3. Read the diskette controller status in BIOS starting with address 40:42. The fifth byte defines the head that the operation ended with. If the operation ended with head 1, the diskette is a 5.25-inch High Capacity (96 TPI) diskette; if the operation ended with head 0, the diskette is a 3.5-inch 1.44M diskette.

Hardware Compatibility

The Personal System/2 products maintain many of the interfaces used by the IBM Personal Computer AT. In most cases command and status organization of these interfaces is maintained.

The functional interfaces for the Model 50 and Model 60 are compatible with the following interfaces:

- The Intel 8259 interrupt controllers (without edge triggering)
- The Intel 8253 timers driven from 1.190 MHz (timer 0 and 2 only)
- The Intel 8237 DMA controller-address/transfer counters, page registers and status fields only. The Command and Request registers are not supported. The rotate and mask functions are not supported. The Mode register is partially supported.
- The NS16450 serial port
- The Intel 8088, 8086, and 80286 microprocessors
- The Intel 8272 diskette drive controller (level-sensitive interrupt)
- The Motorola MC146818 Time of Day Clock command and status (CMOS reorganized)
- The Intel 8042 keyboard port at address hex 0060
- Display modes supported by the IBM Monochrome Display and Printer Adapter, IBM Color/Graphics Monitor Adapter, and the IBM Enhanced Graphics Display Adapter
- The parallel printer ports (Parallel 1, Parallel 2, and Parallel 3) in compatibility mode
- Generally compatible with the Intel 80287 and 8087 math coprocessors (See "8087 to 80287 Compatibility" on page 3-7 for limitations).

Multitasking Provisions

The BIOS contains a feature to assist multitasking implementation. "Hooks" are provided for a multitasking dispatcher. Whenever a busy (wait) loop occurs in the BIOS, a hook is provided for the program to break out of the loop. Also, whenever BIOS services an interrupt, a corresponding wait loop is exited, and another hook is provided. Thus a program can be written that employs the bulk of the device driver code. The following is valid only in the Real Address Mode and must be taken by the code to allow this support.

- The program is responsible for the serialization of access to the device driver. The BIOS code is not reentrant.
- The program is responsible for matching corresponding Wait and Post calls.

Warning: 16-bit operations to the video subsystem can cause a diskette overrun in the 1.44M mode because data width conversions may require more than 12 microseconds. If an overrun occurs, BIOS returns an error code and the operation should be retried.

Interfaces

There are four interfaces to be used by the multitasking dispatcher:

Startup: First, the startup code hooks interrupt hex 15. The dispatcher is responsible to check for function codes of AH = hex 90 or 91. The "Wait" and "Post" sections describe these codes. The dispatcher must pass all other functions to the previous user of interrupt hex 15. This can be done by a JMP or a CALL. If the function code is hex 90 or 91, the dispatcher should do the appropriate processing and return by the IRET instruction.

Serialization: It is up to the multitasking system to ensure that the device driver code is used serially. Multiple entries into the code can result in serious errors.

Wait: Whenever the BIOS is about to enter a busy loop, it first issues an interrupt hex 15 with a function code of hex 90 in AH. This signals a wait condition. At this point, the dispatcher should save the task status and dispatch another task. This allows overlapped execution of tasks when the hardware is busy. The following is an outline of the code that has been added to the BIOS to perform this function.

```
MOV AX, 90XXH          ; wait code in AH and
                       ; type code in AL
INT 15H                ; issue call
JC  TIMEOUT            ; optional: for time-out or
                       ; if carry is set, time-out
                       ; occurred
NORMAL TIMEOUT LOGIC ; normal time-out
```

Post: Whenever the BIOS has set an interrupt flag for a corresponding busy loop, an interrupt hex 15 occurs with a function code of hex 91 in AH. This signals a Post condition. At this point, the dispatcher should set the task status to "ready to run" and return to the interrupt routine. The following is an outline of the code added to BIOS that performs this function.

```
MOV AX, 91XXH          ; post code AH and
                       ; type code AL
INT 15H                ; issue call
```

Classes

The following types of wait loops are supported:

- The class for hex 0 to 7F is serially reusable. This means that for the devices that use these codes, access to the BIOS must be restricted to only one task at a time.
- The class for hex 80 to BF is reentrant. There is no restriction on the number of tasks that can access the device.
- The class for hex C0 to FF is noninterrupt. There is no corresponding interrupt for the wait loop. Therefore, it is the responsibility of the dispatcher to determine what satisfies this condition to exit from the loop.

Function Code Classes

Type Code (AL)	Description
00H->7FH	Serially reusable devices; the operating system must serialize access
80H->0BFH	Reentrant devices; ES:BX is used to distinguish different calls (multiple I/O calls are allowed simultaneously).
0C0H->0FFH	Wait-only calls; there is no complementary Post for these waits--these are time-out only. Times are function-number dependent.

Function Code Assignments: The following are specific assignments for the Model 50 and Model 60 BIOS. Times are approximate. They are grouped according to the classes described under "Function Code Classes."

Type Code (AL)	Time Out	Description
00H	Yes (12 seconds)	Fixed Disk
01H	Yes (2 seconds)	Diskette
02H	No	Keyboard
0FCH	Yes	Fixed Disk Reset
0FDH	Yes (500-ms Read/Write)	Diskette Motor Start
0FEH	Yes (20 seconds)	Printer

Figure 9-3. Functional Code Assignments

The asynchronous support has been omitted. The serial and parallel controllers generate interrupts, but BIOS does not support them in the interrupt mode. Therefore, the support should be included in the multitasking system code if that device is to be supported.

Time-Outs

To support time-outs properly, the multitasking dispatcher must be aware of time. If a device enters a busy loop, it generally should remain there for a specific amount of time before indicating an error. The dispatcher should return to the BIOS wait loop with the carry bit set if a time-out occurs.

Machine-Sensitive Programs

Programs can select machine-specific features, but they must first identify the machine and model type. IBM has defined methods for uniquely determining the specific machine type. The location of the machine model bytes can be found through interrupt 15 function code (AH) = hex C0. The model bytes for Model 50 and Model 60 are shown in the following figure.

Model Byte	Sub-Model Byte	Product Name
FC	04	Model 50
FC	05	Model 60

Figure 9-4. Machine Model Bytes

See the *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference* for a listing of model bytes for other IBM products.