Compatibility

Figures

Description

The differences in system microprocessors, math coprocessors, general system architecture, and diskette drive capabilities must be taken into consideration when designing application programs exclusively for a specific model or programs compatible across the IBM Personal Computer and Personal System/2 product lines. This section discusses these major differences and provides some suggestions to aid you in developing your program.

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-triggered hardware interrupts. The interrupt controller clears its in-service register bit when the interrupt routine sends an End-of-Interrupt (EOI) command to the controller. The EOI command 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 command 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 command 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 command, but prior to enabling the interrupt through the Set Interrupt Enable Flag (STI) command.

In the level-sensitive systems, hardware prevents the interrupt controllers from being set to the edge-triggered 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 EOI command 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 command 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 (for example In, Out, Peek, and Poke) must follow the assembler language rules. See "Assembler Language Programming Considerations."

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 either the 80286 or 80386 microprocessor than they do on systems using the 8088 or 8086 microprocessor.

- PUSH SP

  The 80286 and 80386 microprocessors push 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 and 80386 microprocessors do 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 and 80386 microprocessors push 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 and 80386 microprocessors:

  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.

- LOCK prefix:

  When the LOCK prefix is used with an instruction, the system microprocessor executes the entire instruction before allowing interrupts. If a Repeat String Move instruction is locked, interrupts may be disabled for a long duration.

  The 8088, 8086, and 80286 microprocessors allow the LOCK prefix to be used with most instructions. However, the 80386 microprocessor restricts the use of LOCK to the following instructions:

  - Bit Test and Set Memory, Register/Immediate
  - Bit Test and Reset Memory, Register/Immediate
  - Bit Test and Complement Memory, Register/Immediate
  - XCHG Register, Memory
  - XCHG Memory, Register
  - ADD, OR, ADC, SBB, Memory, Register/Immediate

- AND, SUB, XOR Memory, Register/Immediate
- NOT, NEG, INC, DEC Memory.

An undefined opcode trap (INT 6) is generated if the LOCK prefix is used in the 80386 environment with an instruction not listed.

When the 80286 is operating in the virtual memory mode, the LOCK prefix is IOPL-sensitive. Since the 80386 restricts the use of the LOCK prefix to a specific set of instructions, the LOCK prefix is not IOPL-sensitive in the 80386 environment.

- 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
```

- NT bit and IOPL bits:

  When the 80286 is operating in the Real Address mode, the NT and IOPL bits in the flag register cannot be changed; the bits are zero.

  The 80386 allows the NT bit and the IOPL bits to be modified by POP stack into flags, and other instructions, while operating in the Real Address mode. This has no effect on the Real Address mode operation. However, upon entering Protected Mode operation, the NT bit should be cleared to prevent erroneous execution of the IRET instruction. If NT is set, the IRET attempts to perform a task switch to the previous task.

- Overlap of OUT and following instructions:

  The 80386 has a delayed write to memory and delayed output-to-I/O capability. It is possible for the actual output cycle to I/O devices to occur after the completion of instructions following the Out instruction. Under certain conditions, this may cause some programs to behave in an undesirable manner. For example, an interrupt handler routine may output an EOI command to the interrupt controller to drop the interrupt request. If the interrupt handler has an STI instruction following the output instruction, the 80386 may re-enable interrupts before the interrupt controller drops the interrupt request. This could cause the interrupt routine to be reentered.

  To avoid this problem, either of the following procedures may be used:

  - Place a JMP SHORT $+2 instruction between the OUT instruction and the STI instruction, or
  - Read back the status from the interrupt controller before executing the STI instruction.
  - Math coprocessor instructions:

  In 80386-based systems, the mode of the microprocessor and math coprocessor are tightly coupled. This is not the case for 80286-based systems. The 80286-based systems require the math coprocessor FSETPM instruction to be executed to enable

the 80287 to operate in the Protected mode. The 80287 remains in the Protected mode until it is reset.

The mode of the 80287 determines the format in which the math coprocessor state information is saved by the FSTENV and FSAVE instructions. In the Protected mode, the instruction and data operand pointers are saved as selector/offset pairs; in the Real Address mode, the physical address and opcode are saved.

If the FSETPM instruction is encountered in the 80386 environment, it is ignored. The formatting is performed by the 80386, which internally maintains the instruction and data operand pointers. The Real Address mode format image is saved when the 80386 is operating in the Real Address mode or Virtual 8086 mode. The Protected mode format is used otherwise.

See also "Math Coprocessor Compatibility" on page 20 for more information.

- Use of 32-bit registers and the 32-bit addressing mode:

  It is possible to use the 32-bit registers and 32-bit addressing mode in all operating modes of the 80386 through the use of the operand-size prefix or address-size prefix.

  In a multitasking environment, extreme care must be taken to avoid conflicts with other tasks that use extended registers. If the operating system saves the extended 32-bit registers and new segment registers in the task context save area, conflicts will be avoided; if the operating system does not provide this function, another method must be implemented.

  One possible method is to disable the interrupts while using the extended registers. The extended registers should be saved before use and restored immediately after use while the interrupts are still disabled. The time that interrupts are disabled should be kept as short as possible.

- Operand Alignment:

  When multiple bus cycles are required to transfer a multibyte logical operand (for example, a word operand beginning at an address not evenly divisible by 2), the 80386 transfers the highest order bytes first.

  This characteristic may affect adapters with memory-mapped I/O that require or assume that sequential memory accesses are made to the memory I/O ports.

  This problem may be avoided by using a REP MOVB(yte) instead of a REP MOVSW(ord).

80286 Anomalies

In the Protected mode, when any of the null selector values (hex 0000, 0001, 0002, and 0003) 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 hex 0000 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, an interrupt 9 will not be generated.

The following describes the operation of all 80286 parts:

- Instructions longer than 10 bytes (instructions using multiple redundant prefixes) generate an interrupt 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 interrupt 13.

80386 Anomalies

The following describes anomalies that apply to the B-1 stepping level of the 80386 microprocessor. Use the Interrupt 15 call with (AH) = C9 to determine the stepping level.

80386 Real Address Mode Operation

- FSAVE/FSTENV opcode field incorrect:

  The opcode of some numeric instructions is saved incorrectly in the FSAVE/FSTENV format image when the 80386 is operating in the Real Address mode or Virtual 8086 mode.

  The power-on self-test (POST) code in the system ROM enables hardware interrupt 13 and sets up its vector (INT hex 75) to point to a math coprocessor exception routine in ROM. Any time this routine is executed as a result of an exception, it repairs the opcode field by performing the following sequence:

  1. Clears the 'busy' signal latch
  2. Executes FNSTENV (save image on stack)
  3. Extracts instruction pointer from FSTENV memory image
  4. Skips over prefix bytes until opcode is found
  5. Inserts correct opcode information in the memory image
  6. Executes FLDENV to restore the corrected opcode field

7. Writes the EOI command to the interrupt controller
8. Transfers control to the address pointed to by the NMI handler.

Any math coprocessor application containing an NMI handler should require its NMI handler to read the status of the coprocessor to determine if the NMI was caused by the coprocessor. If the interrupt was not generated by the coprocessor, control should be passed to the original NMI handler.

Applications do not require any modification for this errata because the BIOS exception routine repairs the opcode field after exceptions. However, if a debugger is used to display the math coprocessor state information, the opcode field will contain an incorrect value for some math coprocessor instructions.

- Single stepping repeated MOVS:

  If a repeated MOVS instruction is executed when single-stepping is turned on (TF = 1 in the EFLAGS register), a single-step interrupt is taken after two move steps on the 80386 microprocessor. The 8088, 8086, and 80286 microprocessors take a single-step interrupt after every iteration step. However, for the 80386, if a data breakpoint is encountered on the first iteration of a repeated MOVS, the data break is not taken until after the second iteration. Data breakpoints encountered on the second and subsequent iterations stop immediately after the step causing a break.

- Wrong register size for string instructions:

  One of the (E)CX, (E)SI, or (E)DI registers will not be updated properly if certain string and loop instructions are followed by instructions that either:

  - Use a different address size (that is, either the string instruction or the following instruction uses an address size prefix), or
  - Reference the stack (such as PUSH/POP/CALL/RET) and the "B" bit in the SS descriptor is different from the address size used by the instructions.

  The size of the register (16 bits or 32 bits) is taken from the instruction following the string instruction rather than from the string instruction itself. This could result in one of the following conditions:

- Only the lower 16 bits of a 32-bit instruction updated (if the 32-bit string instruction was followed by an instruction using a memory operand addressed with a 16-bit address).
- All 32 bits of a register updated rather than just the lower 16 bits.

The following is a list of the instructions and the affected registers:

Instruction	Register
REP MOVS	(E)SI
MOVS	(E)DI
STOS	(E)DI
INS	(E)DI
REP INS	(E)CX

Notes:

1. A 32-bit effective address size specified with a string instruction indicates that the 32-bit ESI and EDI registers should be used for forming addresses, and the 32-bit ECX register should be used as the count register.
2. A 32-bit operand size on a repeated string move (MOVS) should be used only if the compiler or programmer can guarantee that the strings do not overlap destructively. An 8-bit or 16-bit MOVS has a predictable effect when the strings overlap destructively.

Figure 1. String Instruction/Register Size Mismatch

The problem only occurs if instructions with different address sizes are mixed, or if a code segment of one size is used with a stack segment of the other size.

To avoid this problem, add a NOP instruction after each of the instructions listed in Figure 1 and ensure that the NOP instruction has the same address size as the string/loop instruction. If necessary, an address size prefix hex 67 may precede the NOP instruction.

- Wrong ECX update with REP INS:

  ECX (or CX in a 16-bit address size) is not updated correctly in the case of a REP INS[1] followed by an early start instruction[2]. After executing any repeat-prefixed instruction, the contents of ECX is supposed to be 0, but in the case of an

---

1 REP INS refers to any input string instruction with a repeat prefix.

2 An early start instruction refers to PUSH, POP, or memory reference instructions.

REP INS instruction, ECX is not updated correctly and its contents become hex FFFFFFFF for 32-bit address size operations and hex 0FFFF for 16-bit address size operations. INS is still executed the correct number of times and EDI is updated properly.

To avoid this problem, one of the following procedures may be used:

- Insert an explicit MOV ECX,0 (or MOV CX,0) instruction after any REP INS instruction. This ensures that the contents of ECX or CX are 0.
- Do not rely on the count in ECX (or CX) after a REP INS instruction but instead, move a new count into ECX (or CX) before using it again.

- Test register access fails:

  Accessing the Translate Lookaside Buffer (TLB) test registers, TR6 and TR7, may not function properly.

  Avoid using test registers TR6 and TR7 to test the TLB.

80286 Compatible Protected Mode Operation

- Math coprocessor Save/Restore environment operands:

  If either of the last two bytes of an FSAVE/FRSTOR or FSTENV/FLDENV is not accessible, the instruction cannot be restarted. An FINIT instruction must be issued to the math coprocessor before any other math coprocessor instruction can be executed. This problem arises only in demand-paged systems, or demand-segmented systems that increase the segment size on demand.

- Wrap-around math coprocessor operands:

  The 80386 architecture does not permit a math coprocessor operand, or any other operand, to wrap around the end of a segment. If such an instruction is issued in a protected segmented system, and the operand starts and ends in valid parts of a segment, but passes through an inaccessible region of the segment, the math coprocessor may be put in an indeterminate state. Under these conditions, an FCLEX or FINIT must be sent before any other math coprocessor instruction is issued.

- Load Segment Limit instruction cannot precede PUSH/POP:

  If the instruction executed immediately after a Load Segment Limit (LSL) instruction does a stack operation, the value of (E)SP may be incorrect after the operation.

  Note: Stack operations resulting from noninstruction sources, such as exceptions or interrupts following the LSL, do not corrupt (E)SP.

  To avoid this problem, make sure that the instruction following an LSL instruction is never one that does a push to or pop off the stack. This includes PUSH, POP, RET, CALL, ENTER, and other such instructions. This can be achieved by always following an LSL instruction with a NOP instruction. Even if a forbidden instruction is used, (E)SP may be updated correctly since the problem is data-dependent and only occurs if the LSL operation succeeds (that is, sets the ZF flag).

- LSL/LAR/VERR/VERW malfunction with a NULL selector:

  An LSL, LAR, VERR, or VERW executed with a NULL selector (that is, bits 15 through 2 of the selector set to 0) operates on the descriptor at entry 0 of the Global Descriptor Table (GDT) instead of unconditionally clearing the ZF flag.

  This problem can be avoided by filling in the "NULL descriptor" (that is, the descriptor at entry 0 of the GDT) with all zeroes, which is an invalid descriptor type.

  The access to the "NULL descriptor" is made but fails since the descriptor has an invalid type. The failure is reported with ZF cleared, which is the desired behavior.

80386 Extended Protected Mode Operation

The following problems exist for operation in the Virtual 8086 mode.

- Task switch to Virtual 8086 mode does not set prefetch limit:

  The 80386 prefetch unit limit is not updated when doing a task switch to the Virtual 8086 mode. This can cause an incorrect segment limit violation to be reported if the microprocessor instruction fetches the segment limit that existed before the task switch.

  This problem can be avoided by using an IRET with the appropriate items on the stack to start the Virtual 8086 task in place of the task switch method.

- FAR jump near page boundary in Virtual 8086 mode:

  When paging is enabled in the Virtual 8086 mode, and a direct FAR jump (opcode EA) instruction is located at the end of a page (or within 16 bytes of the end), and the next page is not cached in the TLB internal to the 80386, the FAR jump instruction leaves the prefetcher limit at the "end" of the old code segment instead of setting it at the "end" of the new code segment. This can allow execution off the end of the new segment to trigger a segment limit violation, or cause a spurious GP fault if the old and new segments overlap and a prefetch crosses the old segment limit.

  There is no way to detect code "walking off" the end of a code segment. However, the spurious GP fault can be avoided by simply performing an IRET back to the instruction causing the fault. The IRET will set the prefetch limit correctly, provided the exception handler has the ability to determine a spurious GP fault from a "real" GP fault.

80486 Anomalies

The following describes anomalies that apply to the B stepping level of the 80486 microprocessor. Use the Interrupt 15 call with (AH) = C9 to determine the stepping level.

Programs with time-delay dependencies should be reexamined for proper operation in systems using the 80486 microprocessor.

Programs using single JMP SHORT $+2 (a delay mechanism) to separate I/O operations will perform properly. However, programs using multiple JMPs to separate I/O operations might not perform properly. Each JMP will delay three clocks on the 80486 microprocessor instead of five clocks as on the 80386 microprocessor.

In protect mode and virtual 8086 mode, the microprocessor allows doubleword accesses to some locations masked by the I/O Permission Bit Map. This problem occurs only when doubleword accesses are made to a port address using an I/O instruction. It does not occur when byte or word accesses are made to I/O ports.

Programs using STI/CLI sequences (interrupt enable/disable) should ensure that multiple instructions execute between the STI and the CLI instructions. A single instruction (including NOP) is not sufficient to guarantee recognition of an interrupt.

80387 Anomalies

FCOMP will return an incorrect comparison when the memory operand is used. When the memory operand is a denormal number with the same exponent as the operand in the ST register but with a different significant part, the comparision indicates equality between the two operands. An alternative method is to put both operands onto the register stack before comparing them. When both operands are on the stack, the comparison result is correct.

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 or 80386 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 1 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 1.44MB 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 ms
High Capacity (96 TPI)	15 ms
3.5-Inch Diskette Drives:	
720KB	20 ms
1.44MB	15 ms

Figure 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 format operation, diskette BIOS enforces a minimum time of 500 milliseconds (1 second for a Personal Computer product). Verify 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 present. 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 1.44MB diskette.

Software Compatibility

To maintain software compatibility, the interrupt polling mechanism used by IBM personal computer products is retained. Software that interfaces with the reset port for the IBM personal computer

positive-edge interrupt sharing[3] does not create interference. Level-sensitive interrupt hardware allows several devices to simultaneously set a common interrupt line active (low) without interference.

Application code that deals directly with the interrupt controller may try to reset the controller to the positive edge-sensitive mode when relinquishing control. The interrupt control circuitry of the system board prevents setting the controller to the edge-sensitive mode by blocking positive edge-sensitive commands to the interrupt controllers.

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 for a reentrant device.
- The program is responsible for matching corresponding Wait and Post calls.

Warning: Because data width conversions can require more than 12 microseconds, 32-bit operations to the video subsystem can cause a diskette overrun in the 1.44MB mode. If an overrun occurs, BIOS returns an error code and the operation should be retried.

3 Hex address 02FX or 06FX, where X is the interrupt level.

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 for checking for function codes of AH= hex 90 or 91. The following "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: The multitasking system must 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, signaling 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 for serially reusable devices. 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 for reentrant devices. There is no restriction on the number of tasks that can access the devices.
- The class for hex C0 to FF is for noninterrupt devices. 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 Personal System/2 BIOS. Times are approximate.

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

Math Coprocessor Compatibility

IBM systems use three math coprocessors: the 8088- and 8086-based systems use the 8087, the 80286-based systems use the 80287, and the 80386-based systems use the 80387.

In the Real Address mode and Virtual 8086 mode, the 80386 computer with an 80387 Math Coprocessor is upward object-code compatible with software for the 8086/8087 and 80286/80287 Real-Address mode systems; in the Protected mode, the 80386/80387 is upward object-code compatible with software for the 80286/80287 Protected-mode systems. However, if a math coprocessor instruction other than FINIT, FSTSW, or FSTCW is executed by an 80386-based system without an 80387 present, the 80386 waits indefinitely for a response from the 80387. This causes the system to stop processing without providing an error indication. To prevent this problem, software should check for the presence of the 80387 before executing math coprocessor instructions. The BIOS equipment function should be used when possible as the method for detecting the presence of the math coprocessor.

The only other differences of operation that may appear when 8086/8087 programs are ported to a Protected-mode 80386/80387 system (**not** using the Virtual 8086 mode), are in the format of operands for the administration instructions FLDENV, FSTEN,

FRSTOR, and FSAVE. These instructions are normally used only by exception handlers and operating systems, not by application programs.

Operating Modes	Software Written for: 8087 Real	80287 Real	80287 Protected
8087 Real Mode	Yes	Yes*	No
80287 Real Mode	Yes*	Yes	No
80387 Real Mode	Yes***	Yes**	No
80387 8086 Virtual Mode	Yes***	Yes**	No
80287 Protected Mode	No	Yes**	Yes
80387 Protected Mode	No	No	Yes**

* See "8087 to 80287 Compatibility."
** See "80287 to 80387 Compatibility."
***See "8087 to 80287 Compatibility" and "80287 to 80387 Compatibility."

Figure 4. Math Coprocessor Software Compatibility

Many changes have been designed into the 80387 to directly support the IEEE standards in hardware. These changes result in increased performance by eliminating the need for software that supports the IEEE standard.

80486 to 80387 Compatibility

The 80486 microprocessor and the level C 80387 coprocessor treat numeric precision exception (PE) differently from previous levels. If the PE bit was reset to 0 before the instruction is executed, the C1(A) bit in the condition code indicates the round-up direction of the last ESC instruction when the result is inexact. If the PE bit was set to 1 before the instruction is executed, the round-up bit is undefined.

The 80486 reports some numeric exceptions later than the level C 80387 does. For some numeric exceptions, the NPX Exception Interrupt (IRQ 13) is not generated until the next noncontrol floating point or FWAIT instruction is about to be executed. On the other hand, the 80387 always generates the NPX Exception Interrupt at the completion of the floating point instruction that caused the exception.

Programs must detect the presence of the microprocessor before using the ET bit in Control Register 0 (CR0). The ET bit in CR0 is hardwired to 1. Programs write 0 or 1 to this bit, but a 1 is always returned on read.

The following problems exist for operations with paging enabled.

- Coprocessor operands:

  To avoid having a nonstartable instruction involving math coprocessor operands in demand-paged systems, ensure the operands do not cross page boundaries. This can be accomplished by aligning math coprocessor operands in 128-byte boundaries within a segment, and aligning the start of segments on 128-byte physical boundaries.

- Page fault error code on stack is not reliable:

  When a page fault (exception 14) occurs, the three defined bits in the error code can be unreliable if a certain sequence of prefetches occurred at the same time.

  Although the page-fault error code pushed onto the page-fault handler stack is sometimes unreliable, the page-fault linear address stored in register CR2 is always correct. The page-fault handler should refer to the page-fault linear address in register CR2 to access the corresponding page table entry and thereby determine whether the page fault was due to a page-not-present condition or a usage violation.

- I/O relocated in paged systems:

  When paging is enabled (PG = 1 in CR0), accessing I/O addresses in the range hex 00001000 to hex 0000FFFF, or accessing a 80387 Math Coprocessor using ESC instructions (I/O addresses hex 800000F8 to hex 800000FF) can generate incorrect I/O addresses on A12 through A31 if the I/O address is the same as a memory linear address that is mapped by the TLB.

  The physical address corresponding to the memory linear address mapped by the TLB is ANDed with the I/O address, causing the I/O address to be incorrect in most cases.

  A suggested method for handling normal I/O addresses between hex 00000 and hex 0FFFF is as follows: The operating system is required to map the lowest (first) 64KB of linear address space to 16 pages, which are defined such that bits 12 through 15 of the linear and physical addresses are equal. This requires that the pages be aligned on a 64KB physical boundary (the physical address associated with the first page has address bits 15 through 0 equal to 0).

  A suggested method for handling the math coprocessor I/O addresses requires that the memory page at linear address hex 80000000 always be

marked "not present" so it cannot be cached in the TLB. This may be accomplished in one of the following ways:

- Require the operating system to handle a 4KB "hole" in the linear address space at the 2GB boundary.
- Restrict the linear address space to a 2GB maximum instead of 4GB. No segments will have a linear address above the 2GB boundary.

- Spurious page level protection fault:

  This problem only occurs when the page table and the directory entries that map the stacks for the inner levels of a task are marked as supervisor access only, and an external bus HOLD comes during the cycle that pops (E)SP off the stack during an inter-level RET or IRET.

  This problem can be avoided by marking the pages that map the inner level stacks (level 0, 1, and 2) to permit the user read access. The segmentation protection mechanism can be used to prevent user access to the linear addresses containing these stacks, if required.

80387 to 80287 Compatibility

The following summarizes the differences between the 80387 and 80287 Math Coprocessors, and provides details showing how 80287 software can be ported to the 80387 Math Coprocessor:

Note: Any migration from 8087 directly to the 80387 must also take into account the differences between the 8087 and the 80287. This information is provided on page 25.

- The 80387 supports only affine closure for infinity arithmetic, not projective closure.
- Operands for FSCALE and FPATAN are no longer restricted in range (except $\pm\infty$); F2XM1 and FPTAN accept a wider range of operands.
- Rounding control is in effect for FLD *constant*.
- Software cannot change entries of the tag word to values (other than empty) that differ from actual register contents.
- In conformance with the IEEE standard, the 80387 does not support special data formats pseudozero, pseudo-NaN, pseudoinfinity, and unnormal.

Exceptions

When the overflow or underflow exception is masked, the only difference from the 80287 is in rounding when overflow or underflow occurs. The 80387 produces results that are consistent with the rounding mode.

For exceptions that are not masked, a number of differences exist due to the IEEE standard and to functional improvements to the architecture of the 80387:

- There are fewer invalid-operation exceptions due to denormal operands, because the instructions FSQRT, FDIV, FPREM, and conversions to BCD or to integer normalize denormal operands before proceeding.
- The FSQRT, FBSTP, and FPREM instructions may cause underflow, because they support denormal operands.
- The denormal exception can occur during the transcendental instructions and the FXTRACT instruction.
- The denormal exception no longer takes precedence over all other exceptions.
- When the operand is zero, the FXTRACT instruction reports a zero-divide exception and leaves $-\infty$ in ST(1).
- The status word has a new bit (SF) that signals when invalid-operation exceptions are due to stack underflow or overflow.
- FLD *extended precision* no longer reports denormal exceptions, because the instruction is not numeric.
- FLD *single/double precision* when the operand is denormal converts the number to extended precision and signals the denormalized operand exception. When loading a signaling NaN, FLD *single/double precision* signals an invalid-operation exception.
- The 80387 only generates quiet NaNs (as on the 80287); however, the 80387 distinguishes between quiet NaNs and signaling NaNs. Signaling NaNs trigger exceptions when they are used as operands; quiet NaNs do not (except for FCOM, FIST, and FBSTP, which also raise IE for quiet NaNs).
- Most 80387 numeric instructions are automatically synchronized by the 80386. No explicit Wait instructions are required for these instructions. To maintain compatibility with systems using the 8087, an explicit Wait is required before each numeric instruction.

- The FLDENV and FRSTOR instructions should be followed by an explicit Wait when used in the 80387 environment. An explicit Wait is not required after these instructions in the 80287 environment.
- The 80287 FSETPM (set Protected mode) instruction performs no useful purpose in the 80387 environment; if encountered, it is ignored.
- The format of the FSAVE and FSTENV instructions is determined by the current mode of the 80386; the Real Address mode format is used when the 80386 is in the Real Address mode, and the Protected mode format is used when the 80386 is in the Protected mode.
- The following applies only to the B1 stepping level 80386: An interrupt 9 does not occur for an operand outside a segment size; an interrupt 13 occurs.

80287 to 8087 Compatibility

The 80287 operating in the Real Address mode can execute 8087 software without major modifications. However, because of differences in the handling of numeric exceptions by the 80287 and the 8087, exception-handling routines *may* need to be changed.

The following summarizes the differences between the 80287 and 8087 Math Coprocessors, and provides details showing how 8087 software can be ported to the 80287 Math Coprocessor.

- The 8087 instructions FENI/FNENI and FDISI/FNDISI perform no useful function in the 80287 environment. If the 80287 encounters one of these opcodes in its instruction stream, the instruction is effectively ignored; none of the 80287 internal states are updated. While 8086 code containing these instructions may be executed on an 80287, it is unlikely that the exception-handling routines containing these instructions will be completely portable to the 80287.
- The ESC instruction address saved in the 80287 includes any leading prefixes before the ESC opcode. The corresponding address saved in the 8087 does not include leading prefixes.
- In the Protected mode, the format of the 80287 saved instruction and address pointers is different from the format of the 8087. The instruction opcode is not saved in the Protected mode; exception handlers have to retrieve the opcode from memory if needed.

- Interrupt 7 occurs in the 80286 when executing ESC instructions with either TS (task switched) or EM (emulation) of the 80286 MSW set (TS = 1 or EM = 1). If TS is set, then a Wait instruction also causes interrupt 7. An exception handler should be included in 80286 code to handle these exceptions.
- Interrupt 9 occurs if the second or subsequent words of a floating-point operand fall outside a segment size. Interrupt 13 occurs if the starting address of a numeric operand falls outside a segment size. An exception handler should be included in the 80286 code to report these programming errors.
- Most 80287 numeric instructions are automatically synchronized by the 80286. The 80286 automatically tests the 'busy' signal from the 80287 to ensure that the 80287 has completed its previous instruction before executing the next ESC instruction. Explicit Wait instructions are not required to ensure this synchronization. An 8087 used with 8086 and 8088 system microprocessors requires explicit Waits before each numeric instruction to ensure synchronization. Although 8086 software having explicit Wait instructions executes perfectly on the 80286 without reassembly, these Wait instructions are unnecessary.

  The processor control instructions for the 80287 may be coded using either a WAIT or No-WAIT form of the mnemonic. The WAIT forms of these instructions cause the assembler to precede the ESC instruction with a microprocessor Wait instruction.

Diskette Drives and Controller

The following figure shows the read, write, and format capabilities for each type of diskette drive.

Diskette Drive Type	160/180KB Mode	320/360KB Mode	720KB Mode	1.44MB 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 5. Diskette Drive Read, Write, and Format Capabilities

Notes:

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

Warning: 32-bit operations to the video subsystem can cause a diskette overrun in the 1.44MB 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 3.5-inch 1.44MB diskette drive.

- Bypassing BIOS Routines:
  - 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.

Detailed information about specific diskette drives is available in separate technical references.

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 system-specific technical references.