IBM

Personal System/2 Hardware Interface Technical Reference

- Common Interfaces

84F9809
S84F-9809-00

IBM

Personal System/2 Hardware Interface Technical Reference - *Common Interfaces*

Personal System/2 Hardware Interface Technical Reference - Common Interfaces

84F9735

First Edition (October 1990)

Special Notices

Hitachi	Hitachi Corporation
Intel	Intel Corporation
Motorola	Motorola, Incorporated

Preface

The Technical Reference library provides hardware and software interface information for IBM Personal System/2 products. The volumes in the library are:

IBM Personal System/2 Hardware Interface Technical Reference – Architectures:
This volume describes the architectures used with Personal System/2 products that are based on the Micro Channel architecture.

IBM Personal System/2 Hardware Interface Technical Reference – Common Interfaces:
This volume describes those devices and interfaces that are common to PS/2 systems. It includes the technical information describing devices, such as the serial port and parallel port controllers, and general information, such as the microprocessor instruction set and the characters associated with each keystroke.

IBM Personal System/2 Hardware Interface Technical Reference – System-Specific Information:
These technical references provides information concerning hardware implementation and performance information for specific models of PS/2 systems.

IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference:
This volume provides BIOS and Advanced BIOS interface information.

Option and Adapter Technical References:
These technical references provide hardware and programming information about individual PS/2 options and adapters.

Suggested Reading:

- *BASIC for the IBM Personal Computer*
- IBM *Disk Operating System (DOS)*
- IBM *Operating System/2*
- *Macro Assembler for the IBM Personal Computer*

Warning: In this technical reference, the term "reserved" is used to describe certain signals, bits, and registers. Use of reserved areas can cause compatibility problems, loss of data, or permanent damage to the hardware.

When modifying a register, the state of the reserved bits must be preserved. When possible, read the register first and change only the bits required.

Note: The vertical bars in the left margin indicate technical changes. The technical changes consist of additions or corrections to the previous information. Only the changes from the last release of that specific technical reference are indicated.

Microprocessors and Instruction Sets

Figures

Notes:

80286 Microprocessor

The 80286 microprocessor subsystem has the following:

- 24-bit address
- 16-bit data interface
- Extensive instruction set, including string I/O
- Hardware fixed-point multiply and divide
- Two operational modes:
  - 8086-compatible Real Address
  - Protected Virtual Address.
- 16MB (MB equals 1,048,576 or 2^{20} bytes) of physical address space
- 1GB (GB equals 1,073,741,824 or 2^{30} bytes) of virtual address space.

Real-Address Mode

In the real-address mode, the address space of the system microprocessor is a contiguous array of up to 1MB. The system microprocessor generates 20-bit physical addresses to address memory.

The segment portion of the pointer is interpreted as the upper 16 bits of a 20-bit segment address; the lower 4 bits are always 0. Therefore, segment addresses begin on multiples of 16 bytes.

All segments in the real-address mode are 64KB (KB equals 1024 bytes) and can be read, written, or executed. An exception or interrupt can occur if data operands or instructions attempt to wrap around the end of a segment (for example, a word with its low-order byte at offset hex FFFF and its high-order byte at hex 0000). If, in the real-address mode, the information contained in the segment does not use the full 64KB, the unused end of the segment can be overlaid by another segment to reduce physical memory requirements.

Protected Virtual Address Mode

The protected virtual address mode (hereafter called protected mode) offers extended physical and virtual memory address space, memory protection mechanisms, and new operations to support operating systems and virtual memory.

The protected mode provides a virtual address space of 1GB for each task mapped into a 16MB physical address space. The virtual address space may be larger than the physical address space, because any use of an address that does not map to a physical memory location will cause a restartable exception.

Like the real-address mode, the protected mode uses 32-bit pointers, consisting of 16-bit selector and offset components. The selector specifies an index into a memory-resident table rather than the upper 16 bits of a real address. The 24-bit base address of the desired segment is obtained from a table in memory. The 16-bit offset is added to the segment base address to form the physical address. The system microprocessor automatically refers to the tables whenever a segment register is loaded with a selector. All instructions that load a segment register refer to the table without additional program support. Each entry in a table is 8-bytes wide.

80287 Math Coprocessor

The optional 80287 Math Coprocessor enables the system to perform high-speed arithmetic, logarithmic, and trigonometric operations. The coprocessor works in parallel with the microprocessor. The parallel operation decreases operating time by allowing the coprocessor to do mathematical calculations while the microprocessor continues to do other functions.

The coprocessor works with seven numeric data types, which are divided into the following three classes:

- Binary integers (three types)
- Decimal integers (one type)
- Real numbers (three types).

Programming Interface

The coprocessor offers extended data types, registers, and instructions to the microprocessor. The coprocessor has eight 80-bit registers, which provide the equivalent capacity of forty 16-bit registers. This register space allows constants and temporary results to be held in registers during calculations, thus reducing memory access, improving speed, and increasing bus availability. The register space can be used as a stack or as a fixed register set. When used as a stack, only the top two stack elements are operated on.

The following figure shows representations of large and small numbers in each data type.

Data Type	Bits	Significant Digits (Decimal)	Approximate Range (Decimal)
Word Integer	16	4	$-32{,}768 \leq x \leq +32{,}767$
Short Integer	32	9	$-2 \times 10^{9} \leq x \leq +2 \times 10^{9}$
Long Integer	64	19	$-9 \times 10^{18} \leq x \leq +9 \times 10^{18}$
Packed Decimal	80	18	$-9..99 \leq x \leq +9..99$ (18 digits)
Short Real *	32	6 - 7	$8.43 \times 10^{-37} \leq x \leq 3.37 \times 10^{38}$
Long Real *	64	15 - 16	$4.19 \times 10^{-307} \leq x \leq 1.67 \times 10^{308}$
Temporary Real **	80	19	$3.4 \times 10^{-4932} \leq x \leq 1.2 \times 10^{4932}$

* The short-real and long-real data types correspond to the single-precision and double-precision data types.

** The temporary-real data type corresponds to the extended-precision data Type.

Figure 1. 80287 Data Types

Hardware Interface

The coprocessor uses the same clock generator as the microprocessor and operates in the asynchronous mode. The coprocessor is wired so that it functions as an I/O device through I/O port addresses hex 00F8, 00FA, and 00FC. The microprocessor sends opcodes and operands through these I/O ports. It also receives and stores results through the same I/O ports. The coprocessor 'busy' signal informs the microprocessor that it is executing; the microprocessor Wait instruction forces the microprocessor to wait until the coprocessor is finished executing.

The coprocessor detects six different exception conditions that can occur during instruction execution:

- Invalid operation
- Denormal operand
- Zero-divide
- Overflow
- Underflow
- Precision.

If the appropriate exception-mask bit within the coprocessor is not set, the coprocessor activates the 'error' signal. The 'error' signal generates a hardware interrupt (IRQ 13) causing the 'busy' signal to be held in the busy state. The 'busy' signal may be cleared by an 8-bit I/O Write command to address hex 00F0, with D7 through D0 equal to 0. This action also clears IRQ 13.

The power-on self-test code in the system ROM enables 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 nonmaskable interrupt (NMI) vector. This maintains code compatibility across the IBM Personal Computer and Personal System/2 product lines. The NMI handler reads the coprocessor status to determine if the coprocessor generated the NMI. If it was not generated by the coprocessor, control is passed to the original NMI handler.

The coprocessor has two operating modes: real-address mode and protected mode. They are similar to the two modes of the microprocessor. The coprocessor is in the real-address mode if reset by a power-on reset, system reset, or I/O write operation to port hex 00F1. This mode is compatible with the 8087 Math Coprocessor used in IBM Personal Computers. The coprocessor is placed in the protected mode by executing the SETPM ESC instruction. It is placed back in the real-address mode by an I/O write operation to port hex 00F1, with D7 through D0 equal to 0.

Detailed information for the internal functions of the 80287 Math Coprocessor is in the books listed in the Bibliography. Also see "Compatibility" for more information.

80386 Microprocessor

The 80386 microprocessor subsystem has the following:

- 32-bit address
- 32-bit data interface
- Extensive instruction set, including string I/O
- Hardware fixed-point multiply and divide
- Three operational modes:
  - Real Address
  - Protected Virtual Address
  - Virtual 8086.

- 4GB of physical address space
- 8 general-purpose 32-bit registers
- 64TB (TB equals 1,099,511,627,776 or 2^{40} bytes) of total virtual-address space.

Real Address Mode

In the real-address mode, the address space of the system microprocessor is a contiguous array of up to 1MB. The system microprocessor generates 20-bit physical addresses to address memory.

The segment portion of the pointer is interpreted as the upper 16 bits of a 20-bit segment address; the lower 4 bits are always 0. Therefore, segment addresses begin on multiples of 16 bytes.

All segments in the real-address mode are 64KB and can be read, written, or executed. An exception or interrupt can occur if data operands or instructions attempt to wrap around the end of a segment (for example, a word with its low-order byte at offset hex FFFF and its high-order byte at hex 0000). If, in the real-address mode, the information contained in the segment does not use the full 64KB, the unused end of the segment can be overlaid by another segment to reduce physical memory requirements.

Protected Virtual Address Mode

The protected virtual-address mode offers extended physical and virtual memory address space, memory protection mechanisms, and new operations to support operating systems and virtual memory.

The protected mode provides up to 64TB of virtual address space for each task mapped into a 4GB physical address space.

From a programmer's point of view, the main difference between the real-address mode and protected mode is the increased address space and the method of calculating the base address. The protected mode uses 32- or 48-bit pointers, consisting of 16-bit selector and 16- or 32-bit offset components. The selector specifies an index into one of two memory-resident tables, the global descriptor table (GDT) or the local descriptor table (LDT). These tables contain the 32-bit base address of a given segment. The 32-bit effective offset is added to the segment base address to form the physical address. The system microprocessor automatically refers to the tables whenever a segment register is loaded with a selector. All instructions that load a segment register refer to the memory-resident tables without additional program support. The memory-resident tables contain 8-byte values called descriptors.

The paging option provides an additional way of managing memory in the very large segments of the 80386. Paging operates in the protected mode only, beneath segmentation. The paging mechanism translates the protected linear address (which comes from the segmentation unit) into a physical address. When paging is not enabled, the physical address is the same as the linear address. The following figure shows the 80386 addressing mechanism.

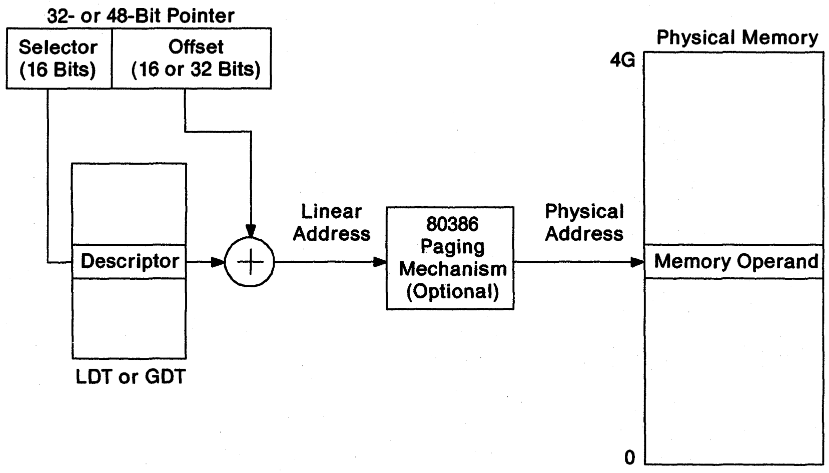


Figure 2. 80386 Addressing

Virtual 8086 Mode

The virtual-8086 mode ensures compatibility of programs written for 8086- and 8088-based systems by establishing a protected 8086 environment within the 80386 multitasking framework.

Since the address space of an 8086 is limited to 1MB, the logical addresses generated by the virtual-8086 mode lie within the first 1MB of the 80386 linear address space. To support multiple virtual-8086 tasks, paging can be used to give each virtual-8086 task a 1MB address space anywhere in the 80386 physical address space.

On a task-by-task basis, the value of the virtual-8086 flag (VM86 flag in the Flags register) determines whether the 80386 behaves as an 80386 or as an 8086. Some instructions, such as Clear Interrupt Flag, can disrupt all operations in a multitasking environment. The 80386 raises an exception when a virtual-8086 mode task attempts to execute an I/O instruction, interrupt-related instruction, or other sensitive instruction. Anytime an exception or interrupt occurs, the 80386 leaves the virtual 8086 mode, making the full resources of the 80386 available to an interrupt handler or exception handler. These handlers can determine if the source of the exception was a virtual-8086 mode task by inspecting the VM86 flag in the Flags image on the stack. If the source is a virtual-8086 mode task, the handler calls on a routine in the operating system to simulate an 8086 instruction and return to the virtual-8086 mode.[1]

80386 Paging Mechanism

The 80386 uses two levels of tables to translate the linear address from the segmentation unit into a physical address. There are three components to the paging mechanism:

- Page directory
- Page tables
- Page frame (the page itself).

The figure on the following page shows how the two-level paging mechanism works.

1 The routine in the operating system, called a *virtual machine monitor*, simulates a limited number of 8086 instructions.

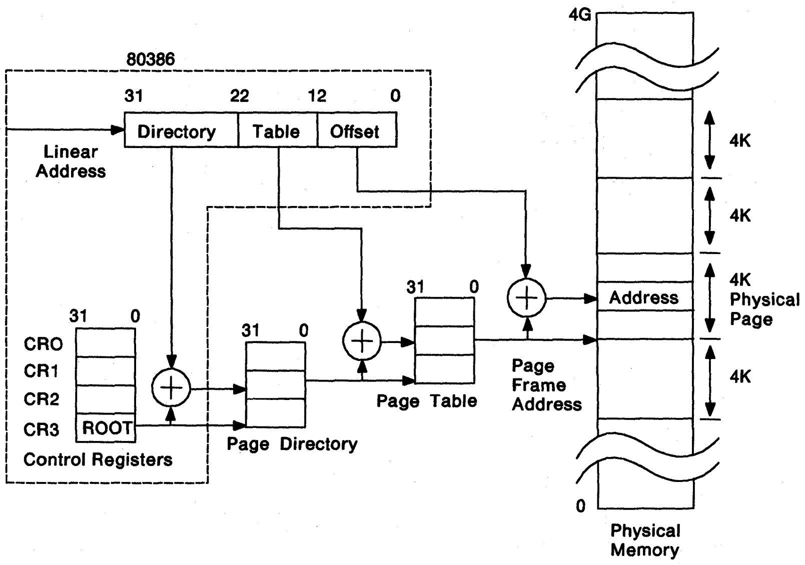


Figure 3. Paging Mechanism

CR2 is the Page-Fault Linear-Address register. It holds the 32-bit linear address that caused the last detected page fault.

CR3 is the Page Directory Physical Base Address register. It contains the physical starting address of the page directory.

The page directory is 4KB and allows up to 1024 page-directory entries. Each page-directory entry contains the address of the next level of tables, the page tables, and information about the page tables. The upper 10 bits of the linear address (A22 through A31) are used as an index to select the correct page-directory entry.

Each page table is 4KB and holds up to 1024 page-table entries. Page-table entries contain the starting address of the page frame and statistical information about the page. Address bits A12 through A21 are used as an index to select one of the 1024 page-table entries. The upper 20 bits of the page-frame address (from the page-table entry) are linked with the lower 12 bits of the linear address to form the physical address. The page-frame address bits become the most-significant bits; the linear-address bits become the least-significant bits.

80387 Math Coprocessor

The optional 80387 Math Coprocessor enables the system to perform high-speed arithmetic, logarithmic, and trigonometric operations. The 80387 effectively extends the 80386 register and instruction set for existing data types and also adds several new data types. The following figure shows the four data type classifications and the instructions associated with each.

Classification	Size	Instructions
Integer	16, 32, 64 Bits	Load, Store, Compare, Add, Subtract, Multiply, Divide
Packed BCD*	80 Bits	Load, Store
Real	32, 64 Bits	Load, Store, Compare, Add, Subtract, Multiply, Divide
Temporary Real	80 Bits	Add, Subtract, Multiply, Divide, Square Root, Scale, Remainder, Integer Part, Change Sign, Absolute Value, Extract Exponent and Significand, Compare, Examine, Test, Exchange Tangent, Arctangent, $2^x - 1$, $Y*Log_2 (X+1)$, $Y*Log_2 (X)$, Load Constant (0.0, π, etc.), Sine, Cosine, Unordered Compare

* BCD = Binary-coded decimal

Figure 4. Data Type Classifications and Instructions

The 80386/80387 configuration fully conforms to the ANSI[2] and IEEE[3] floating-point standard and are upward, object-code compatible from 80286/80287- and 8086/8087-based systems.

2 American National Standards Institute

3 Institute of Electrical and Electronics Engineers

80387 To 80486 Math Coprocessor Compatibility

The 80387 floating-point coprocessor is integrated into the 80486 microprocessor. All numeric 80387 instructions are fully compatible with the 80486 floating-point unit. The 80486 microprocessor supports the 80486 floating-point error reporting modes to ensure DOS compatibility with 80386/80387 systems.

The coprocessor presence test will always show the presence of a coprocessor in the 80486.

Programs for the 80386/80387 systems that explicitly reset the coprocessor by writing to hex 00F1 will no longer function because the coprocessor is an integral part of the microprocessor. Coprocessor reset or initialization must be accomplished through FINIT/FSAVE.

For DOS compatibility, the numeric exception bit Control Register 0 must be set to 0.

Programming Interface

The 80387 is not sensitive to the processing mode of the 80386. The 80387 functions the same whether the 80386 is executing in real-address mode, protected mode, or virtual-8086 mode. All memory access is handled by the 80386; the 80387 merely operates on instructions and values passed to it by the 80386.

All communication between the 80386 and 80387 is transparent to application programs. The 80386 automatically controls the 80387 whenever a numeric instruction is executed. All physical and virtual memory is available for storage of instructions and operands of programs that use the 80387. All memory address modes, including use of displacement, base register, index register, and scaling are available for addressing numeric operands.

The coprocessor has eight 80-bit registers. The total capacity of these eight registers is equivalent to twenty 32-bit registers. This register space allows constants and temporary results to be held in registers during calculations, thus reducing memory access, improving speed, and increasing bus availability. The register space can be used as a stack or as a fixed register set. When it is used as a stack, only the top two stack elements are operated on.

The following figure shows the seven data types supported by the 80387 Math Coprocessor.

Data Type	Range	Precision
Word Integer	10^4	16 Bits
Short Integer	10^9	32 Bits
Long Integer	10^{19}	64 Bits
Packed BCD	10^{18}	18 Digits (2 digits per byte)
Single Precision (Short Real)	$10^{\pm 38}$	24 Bits
Double Precision (Long Real)	$10^{\pm 308}$	53 Bits
Extended Precision (Temporary Real)	$10^{\pm 4932}$	64 Bits

Figure 5. 80387 Data Types

Hardware Interface

The 80387 Math Coprocessor uses the same clock generator as the 80386 system microprocessor. The coprocessor is wired so that it functions as an I/O device through I/O port addresses hex 00F8, 00FA, and 00FC. The system microprocessor sends opcodes and operands through these I/O ports. The coprocessor 'busy' signal informs the system microprocessor that it is executing an instruction; the system microprocessor Wait instruction forces the system microprocessor to wait until the coprocessor is finished executing the instruction.

The coprocessor detects six different exception conditions that can occur during instruction execution:

- Invalid operation
- Denormal operand
- Zero-divide
- Overflow
- Underflow
- Precision.

If the appropriate exception mask bit within the coprocessor is not set, the coprocessor activates the 'error' signal. The 'error' signal generates a hardware interrupt (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 D7 through D0 equal to 0. This action also clears IRQ 13.

The power-on self-test code in the system ROM enables 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 coprocessor generated the NMI. If it was not generated by the coprocessor, control is passed to the original NMI handler.

Detailed information about the internal functions of the 80387 Math Coprocessor is in the books listed in the Bibliography. Also see "Compatibility" for more information.

80486 Microprocessor

The 80486 microprocessor subsystem has the following:

- 32-bit address
- 32-bit data interface
- Extensive instruction set, including string I/O
- Hardware fixed-point multiply and divide
- Three operational modes:
  - Real Address
  - Protected Virtual Address
  - Virtual 8086
- 4GB of physical address space
- 8 general-purpose 32-bit registers
- 64TB of total virtual-address space
- Internal 8KB, set-associative cache with controller
- Internal 80387 coprocessor.

The 80486 microprocessor is compatible with the 80386 in the following areas:

- Real Address Mode
- Protected Virtual Address Mode
- Virtual 8086 Mode
- 80386 Paging Mechanism
- All published 80386 instructions
- All published 80387 instructions.

The complete 80387 Math Coprocessor instruction set and register set have been included in the 80486 as a floating-point unit. No I/O cycles are executed during floating-point instructions. The 80486 microprocessor is 80386/80387 compatible except for resets to the floating-point unit. Software must use FINIT/FSAVE to reset the floating-point unit (math coprocessor). The instruction and data pointers are set to zero after FINIT/FSAVE.

Cache Control

The 80486 microprocessor contains an 8KB integrated cache for code and data. The cache is managed in two ways, and the operation of the cache has no effect on the operation of any program.

The cache is managed by bit 30 — Cache Disable (CD) and bit 29 — Not Write Through (NW) in Control Register 0 (CR0):

Bit 30 CD	Bit 29 NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidate disabled
1	0	Cache fills disabled, write-through and invalidate enabled
0	1	Reserved
0	0	Cache fills enabled, write-through and invalidate enabled (Normal operating mode)

Figure 6. Control Register 0

Cache Paging Control

The page-write-through (PWT) bit and the page-cache-disabled (PCD) bit are two new bits defined in entries in both levels of the page table structure, the page-directory table and the page-table entry, and in Control Register 3.

The PWT bit (bit 4) controls cache write policy. When this bit is set to 1, a write-through policy for the current 4KB page is defined. When this bit is set to 0, it allows the possibility of write-back policy. This bit is ignored internally because the 80486 microprocessor has a write-through-only cache. The PWT bit can be used to control the write policy of a second-level (external) cache.

The PCD bit (bit 3),in conjunction with the KEN# (cache enabled) input signal and the cache-enable and write-transparent bits in Control Register 0 (CR0), controls the ability of cache. When this bit is set to 1, caching is disabled for the 4KB page regardless of the KEN#, cache-enable bit, and write-through bit. These two bits are also driven external to the processor during memory access to manage a second-level cache, if one exists.

The page-write-through and page-cache-disable bits for a bus cycle are obtained either from Control Register 3, the page-directory entry, or the page-table entry, depending on the type of cycle performed.

Page Protection Feature

The 80486 microprocessor has a new protection feature. The write-protect (WP) bit in CR0 has been added to the 80486 microprocessor to protect read-only pages from supervisor write accesses. The 80386 microprocessor allows a read-only page to be written from protection level 0, 1, or 2. When the WP bit is set to 0, the 80486 microprocessor is in the 80386-compatible mode. When the WP bit is set to 0, the supervisor write access to a read-only page (Read/Write is set to 0) causes a page fault (exception 14).

The write-protect bit has a new feature. This feature involves the use of three new bits in CR0:

- User/Supervisor – U/S
- Read/Write – R/W
- Write/Protect – WP.

The compatible protection feature is described by the following table.

U/S	R/W	WP	User Access	Supervisor Access
0	0	0	None	Read/Write/Execute
0	1	0	None	Read/Write/Execute
1	0	0	Read/Execute	Read/Write/Execute
1	1	0	Read/Write/Execute	Read/Write/Execute

Figure 7. 80386 Compatible Operation

The new protection feature is given by the following table.

U/S	R/W	WP	User Access	Supervisor Access
0	0	1	None	Read/Execute
0	1	1	None	Read/Write/Execute
1	0	1	Read/Execute	Read/Execute
1	1	1	Read/Write/Execute	Read/Write/Execute

Figure 8. 80486 Protection Operation

New Alignment Check

The Flag register in the 80486 microprocessor contains a new bit not available in the 80386. The new bit, alignment check, is bit 18 of the Flag register and enables fault reporting on accesses to misaligned data (through interrupt 17 with an error code 0).

When alignment check is set to 1, it enables fault reporting if memory reference is to a misaligned address. A misaligned address is a word access to an odd address, a doubleword access to an address not on a doubleword boundary, or an 8-byte reference to an address that is not on a 64-bit boundary.

Alignment faults are generated only by a program running at privilege level 3. The alignment-check bit is ignored at privilege levels 0, 1, and 2.

The alignment-check bit is conditioned by a new alignment mask bit, defined as bit 18 in Control Register 0. The alignment-mask bit controls whether the alignment-check bit in the Flag register can allow an alignment fault. When the alignment-mask bit is set to 0, the alignment-check bit is disabled and compatible with the 80386 microprocessor. When the alignment-mask bit is set to 1, the alignment-check bit is enabled.

New Instructions

In addition, the 80486 has six unique instructions that control cache operation:

- Byte Swap (BSWAP)
- Compare and Exchange (CMPXCHG)
- Exchange-and-Add (XADD)
- Invalidate Data Cache (INVD)
- Invalidate TLBN Entry (INVLPG).
- Write-Back and Invalidate Data Cache (WBINVD).

80286 Microprocessor Instruction Set

Data Transfer

MOV = Move

Register to Register/Memory

1 0 0 0 1 0 0 w	mod reg r/m

Register/Memory to Register

1 0 0 0 1 0 1 w	mod reg r/m

Immediate to Register/Memory

1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1

Immediate to Register

1 0 1 1 wreg	data	data if w = 1

Memory to Accumulator

1 0 1 0 0 0 0 w	addr-low	addr-high

Accumulator to Memory

1 0 1 0 0 0 1 w	addr-low	addr-high

Register/Memory to Segment Register

1 0 0 0 1 1 1 0	mod 0 reg r/m

Segment Register to Register/Memory

1 0 0 0 1 1 0 0	mod 0 reg r/m

PUSH = Push

Memory

1 1 1 1 1 1 1 1	mod 1 1 0 r/w

Register

0 1 0 1 0 reg

Segment Register

0 0 0 reg 1 1 0

Immediate

0 1 1 0 1 0 s 0	data	data if s = 0

PUSHA = Push All

0 1 1 0 0 0 0 0

POP = Pop

Register/Memory

1 0 0 0 1 1 1 1	mod 0 0 0 r/m

Register

0 1 0 1 1 reg

Segment Register

0 0 0 reg 1 1 1	reg ≠ 0 1

POPA = Pop All

0 1 1 0 0 0 0 1

XCHG = Exchange

Register/Memory with Register

1 0 0 0 0 1 1 w	mod reg r/m

Register with Accumulator

1 0 0 1 0 reg

IN = Input From

Fixed Port

1 1 1 0 0 1 0 w	port

Variable Port

1 1 1 0 1 1 0 w

OUT = Output To

Fixed Port

1 1 1 0 0 1 1 w	port

Variable Port

1 1 1 0 1 1 1 w

XLAT = Translate Byte to AL

1 1 0 1 0 1 1 1

LEA = Load EA to Register

1 0 0 0 1 1 0 1	mod reg r/m

LDS = Load Pointer to DS

1 1 0 0 0 1 0 1	mod reg r/m mod $\neq$ 1 1

LES = Load Pointer to ES

1 1 0 0 0 1 0 0	mod reg r/m mod $\neq$ 1 1

LAHF = Load AH with Flags

1 0 0 1 1 1 1 1

SAHF = Store AH with Flags

1 0 0 1 1 1 1 0

PUSHF = Push Flags

1 0 0 1 1 1 0 0

POPF = Pop Flags

1 0 0 1 1 1 0 1

Arithmetic

ADD = Add

Register/Memory with Register to Either

0 0 0 0 0 0 dw	mod reg r/m

Immediate to Register/Memory

1 0 0 0 0 0 sw	mod 0 0 0 r/m	data	data if sw = 0 1

Immediate to Accumulator

0 0 0 0 0 1 0 w	data	data if w = 1

ADC = Add with Carry

Register/Memory with Register to Either

0 0 0 1 0 0 dw	mod reg r/m

Immediate to Register/Memory

1 0 0 0 0 0 sw	mod 0 1 0 r/m	data	data if sw = 0 1

Immediate to Accumulator

0 0 0 1 0 1 0 w	data	data if w = 1

INC = Increment

Register/Memory

1 1 1 1 1 1 1 w	mod 0 0 0 r/m

Register

0 1 0 0 0 reg

SUB = Subtract

Register/Memory with Register to Either

0 0 1 0 1 0 dw	mod reg r/m

Immediate from Register/Memory

1 0 0 0 0 0 sw	mod 1 0 1 r/m	data	data if sw = 0 1

Immediate from Accumulator

0 0 1 0 1 1 0 w	data	data if w = 1

SBB = Subtract with Borrow

Register/Memory with Register to Either

0 0 0 1 1 0 dw	mod reg r/m

Immediate from Register/Memory

1 0 0 0 0 0 sw	mod 0 1 1 r/m	data	data if sw = 0 1

Immediate from Accumulator

0 0 0 1 1 1 0 w	data	data if w = 1

DEC = Decrement

Register/Memory

1 1 1 1 1 1 1 w	mod 0 0 1 r/m

Register

0 1 0 0 1 reg

MP = Compare

egister/Memory with Register

0 0 1 1 1 0 1 w	mod reg r/m

Register with Register/Memory

0 0 1 1 1 0 0 w	mod reg r/m

Immediate with Register/Memory

1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if sw = 0 1

Immediate with Accumulator

0 0 1 1 1 1 0 w	data	data if w = 1

NEG = Change Sign

1 1 1 1 0 1 1 w	mod 0 1 1 r/m

AAA = ASCII Adjust for Add

0 0 1 1 0 1 1 1

DAA = Decimal Adjust for Add

0 0 1 0 0 1 1 1

AAS = ASCII Adjust for Subtract

0 0 1 1 1 1 1 1

DAS = Decimal Adjust for Subtract

0 0 1 0 1 1 1 1

MUL = Multiply (Unsigned)

1 1 1 1 0 1 1 w	mod 1 0 0 r/m

IMUL = Integer Multiply (Signed)

1 1 1 1 0 1 1 w	mod 1 0 1 r/m

IIMUL = Integer Immediate Multiply (Signed)

0 1 1 0 1 0 s 1	mod reg r/m	data	data if s = 0

DIV = Divide (Unsigned)

1 1 1 1 0 1 1 w	mod 1 1 0 r/m

IDIV = Integer Divide (Signed)

1 1 1 1 0 1 1 w	mod 1 1 1 r/m

AAM = ASCII Adjust for Multiply

1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0

AAD = ASCII Adjust for Divide

1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0

CBW = Convert Byte to Word

1 0 0 1 1 0 0 0

CWD = Convert Word to Doubleword

1 0 0 1 1 0 0 1

Logic

Shift/Rotate Instructions

Register/Memory by 1

1 1 0 1 0 0 0 w	mod T T T r/m

Register/Memory by CL

1 1 0 1 0 0 1 w	mod T T T r/m

Register/Memory by Count

1 1 0 0 0 0 0 w	mod T T T r/m	count

T T T	**Instruction**
0 0 0	ROL
0 0 1	ROR
0 1 0	RCL
0 1 1	RCR
1 0 0	SHL/SAL
1 0 1	SHR
1 1 1	SAR

AND = And

Register/Memory and Register to Either

0 0 1 0 0 0 dw	mod reg r/m

Immediate to Register/Memory

1 0 0 0 0 0 0 w	mod 100 r/m	data	data if w = 1

Immediate to Accumulator

0 0 1 0 0 1 0 w	data	data if w = 1

TEST = AND Function to Flags; No Result

Register/Memory and Register

1 0 0 0 0 1 0 w	mod reg r/m

Immediate Data and Register/Memory

1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1

Immediate Data and Accumulator

1 0 1 0 1 0 0 w	data	data if w = 1

Or = Or

Register/Memory and Register to Either

0 0 0 0 1 0 d w	mod reg r/m

Immediate to Register/Memory

1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1

Immediate to Accumulator

0 0 0 0 1 1 0 w	data	data if w = 1

XOR = Exclusive OR

Register/Memory and Register to Either

0 0 1 1 0 0 d w	mod reg r/m

Immediate to Register/Memory

1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1

Immediate to Accumulator

0 0 1 1 0 1 0 w	data	data if w = 1

NOT = Invert Register/Memory

1 1 1 1 0 1 1 w	mod 0 1 0 r/m

String Manipulation

MOVS = Move Byte Word

1 0 1 0 0 1 0 w

CMPS B/W = Compare Byte/Word

1 0 1 0 0 1 1 w

SCAS = Scan Byte/Word

1 0 1 0 1 1 1 w

LODS = Load Byte/Word to AL/AX

1 0 1 0 1 1 0 w

STOS = Store Byte/Word from AL/AX

1 0 1 0 1 0 1 w

INS = Input Byte/Word from DX Port

0 1 1 0 1 1 0 w

OUTS = Output Byte/Word to DX Port

0 1 1 0 1 1 1 w

REP/REPNE, REPZ/REPNZ = Repeat String

Repeat Move String

1 1 1 1 0 0 1 1	1 0 1 0 0 1 0 w

Repeat Compare String (z/Not z)

1 1 1 1 0 0 1 z	1 0 1 0 0 1 1 w

Repeat Scan String (z/Not z)

1 1 1 1 0 0 1 z	1 0 1 0 1 1 1 w

Repeat Load String

1 1 1 1 0 0 1 1	1 0 1 0 1 1 0 w

Repeat Store String

1 1 1 1 0 0 1 1	1 0 1 0 1 0 1 w

Repeat Input String

1 1 1 1 0 0 1 1	0 1 1 0 1 1 0 w

Repeat Output String

1 1 1 1 0 0 1 1	0 1 1 0 1 1 1 w

Control Transfer

CALL = Call

Direct within Segment

1 1 1 0 1 0 0 0	disp-low	disp-high

Register/Memory Indirect within Segment

1 1 1 1 1 1 1 1	mod 0 1 0 r/m

Direct Intersegment

1 0 0 1 1 0 1 0	Segment Offset	Segment Selector

Indirect Intersegment

1 1 1 1 1 1 1 1	mod 0 1 1 r/m (mod $\neq$ 11)

JMP = Unconditional Jump

Short/Long

1 1 1 0 1 0 1 1	disp-low

Direct within Segment

1 1 1 0 1 0 0 1	disp-low	disp-high

Register/Memory Indirect within Segment

1 1 1 1 1 1 1 1	mod 1 0 0 r/m

Direct Intersegment

1 1 1 0 1 0 1 0	Segment Offset	Segment Selector

Indirect Intersegment

1 1 1 1 1 1 1 1	mod 1 0 1 r/m (mod ≠ 11)

RET = Return from Call

Within Segment

1 1 0 0 0 0 1 1

Within Segment Adding Immediate to SP

1 1 0 0 0 0 1 0	data-low	data-high

Intersegment

1 1 0 0 1 0 1 1

Intersegment Adding Immediate to SP

1 1 0 0 1 0 1 0	data-low	data-high

JE/JZ = Jump on Equal/Zero

0 1 1 1 0 1 0 0	disp

JL/JNGE = Jump on Less/Not Greater, or Equal

0 1 1 1 1 1 0 0	disp

JLE/JNG = Jump on Less, or Equal/Not Greater

0 1 1 1 1 1 1 0	disp

JB/JNAE = Jump on Below/Not Above, or Equal

0 1 1 1 0 0 1 0	disp

JBE/JNA = Jump on Below, or Equal/Not Above

01110110	disp

JP/JPE = Jump on Parity/Parity Even

01111010	disp

JO = Jump on Overflow

01110000	disp

JS = Jump on Sign

01111000	disp

JNE/JNZ = Jump on Not Equal/Not Zero

01110101	disp

JNL/JGE = Jump on Not Less/Greater, or Equal

01111101	disp

JNLE/JG = Jump on Not Less, or Equal/Greater

01111111	disp

JNB/JAE = Jump on Not Below/Above, or Equal

01110011	disp

JNBE/JA = Jump on Not Below, or Equal/Above

01110111	disp

JNP/JPO = Jump on Not Parity/Parity Odd

0 1 1 1 1 0 1 1	disp

JNO = Jump on Not Overflow

0 1 1 1 0 0 0 1	disp

JNS = Jump on Not Sign

0 1 1 1 1 0 0 1	disp

LOOP = Loop CX Times

1 1 1 0 0 0 1 0	disp

LOOPZ/LOOPE = Loop while Zero/Equal

1 1 1 0 0 0 0 1	disp

LOOPNZ/LOOPNE = Loop while Not Zero/Not Equal

1 1 1 0 0 0 0 0	disp

JCXZ = Jump on CX Zero

1 1 1 0 0 0 1 1	disp

ENTER = Enter Procedure

1 1 0 0 1 0 0 0	data-low	data-high

LEAVE = Leave Procedure

1 1 0 0 1 0 0 1

INT = Interrupt

Type Specified

11001101

Type 3

11001100

INTO = Interrupt on Overflow

11001110

IRET = Interrupt Return

11001111

BOUND = Detect Value Out of Range

01100010	mod reg r/m

Processor Control

CLC = Clear Carry

11111000

CMC = Complement Carry

11110101

STC = Set Carry

11111001

CLD = Clear Direction

11111100

STD = Set Direction

11111101

CLI = Clear Interrupt

11111010

STI = Set Interrupt Enable Flag

11111011

HLT = Halt

11110100

WAIT = Wait

10011011

LOCK = Bus Lock Prefix

11110000

CTS = Clear Task Switched Flag

00001111	00000110

ESC = Processor Extension Escape

11011TTT	mod LLL r/m

Protection Control

LGDT = Load Global Descriptor Table Register

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 1 0 r/m

SGDT = Store Global Descriptor Table Register

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 0 0 r/m

LIDT = Load Interrupt Descriptor Table Register

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 1 1 r/m

SIDT = Store Interrupt Descriptor Table Register

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 0 1 r/m

LLDT = Load Local Descriptor Table Register from Register/Memory

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 1 0 r/m

SLDT = Store Local Descriptor Table Register from Register/Memory

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 0 r/m

LTR = Load Task Register from Register/Memory

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 1 1 r/m

STR = Store Task Register to Register/Memory

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 1 r/m

LMSW = Load Machine Status Word from Register/Memory

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 1 1 0 r/m

SMSW = Store Machine Status Word

00001111	00000001	mod 1 0 0 r/m

LAR = Load Access Rights from Register/Memory

00001111	00000010	mod reg r/m

LSL = Load Segment Limit from Register/Memory

00001111	00000011	mod reg r/m

ARPL = Adjust Requested Privilege Level from Register/Memory

01100011	mod reg r/m

VERR = Verify Read Access; Register/Memory

00001111	00000000	mod 1 0 0 r/m

VERW = Verify Write Access

00001111	00000000	mod 1 0 1 r/m

The effective address (EA) of the memory operand is computed according to the mod and r/m fields:

If mod = 11, then r/m is treated as a reg field.
If mod = 00, then disp = 0, disp-low and disp-high are absent.
If mod = 01, then disp = disp-low sign-extended to 16 bits, disp-high is absent.
If mod = 10, then disp = disp-high:disp-low.

If r/m = 000, then EA = (BX) + (SI) + DISP
If r/m = 001, then EA = (BX) + (DI) + DISP
If r/m = 010, then EA = (BP) + (SI) + DISP
If r/m = 011, then EA = (BP) + (DI) + DISP
If r/m = 100, then EA = (SI) + DISP
If r/m = 101, then EA = (DI) + DISP
If r/m = 110, then EA = (BP) + DISP
If r/m = 111, then EA = (BX) + DISP

The disp field follows the second byte of the instruction (before data if required).

Note: An exception to the above statements occurs when mod = 00 and r/m = 110, in which case EA = disp-high; disp-low.

Segment Override Prefix

0 0 1 reg 1 1 0

The 2-bit and 3-bit reg fields are defined in the following figures.

Reg	**Segment Register**	**Reg**	**Segment Register**
00	ES	10	SS
01	CS	11	DS

Figure 9. 2-Bit Register Field

Figure 10. 3-Bit Register Field

16-Bit (w = 1)	**8-Bit (w = 0)**
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS Segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

80287 Math Coprocessor Instruction Set

The following is an instruction-set summary for the 80287 Math Coprocessor.

The following figure shows abbreviations used in the summary.

Field	Description	Bit Information
escape	80286 Extension Escape	Bit Pattern = 11011
MF	Memory Format	00 = 32-Bit Real 01 = 32-Bit Integer 10 = 64-Bit Real 11 = 16-Bit Integer
ST(0)	Current Stack Top	
ST(i)	ith Register Below the Stack Top	
d	Destination	0 = Destination is ST(0) 1 = Destination is ST(i)
P	Pop	0 = No pop 1 = Pop ST(0)
R	Reverse*	0 = Destination (op) source 1 = Source (op) destination
* When d = 1, reverse the sense of R.		

Figure 11. 80287 Encoding Field Summary

Data Transfer

FLD = Load

Integer/Real Memory to ST(0)

escape MF 1

Long Integer Memory to ST(0)

escape 1 1 1	mod 1 0 1 r/m

Temporary Real Memory to ST(0)

escape 0 1 1	mod 1 0 1 r/m

BCD Memory to ST(0)

escape 1 1 1	mod 1 0 0 r/m

ST(i) to ST(0)

escape 0 0 1	1 1 0 0 0 ST(i)

FST = Store

ST(0) to Integer/Real Memory

escape MF 1	mod 0 1 0 r/m

ST(0) to ST(i)

escape 1 0 1	1 1 0 1 0 ST(i)

FSTP = Store and Pop

ST(0) to Integer/Real Memory

escape MF 1	mod 0 1 1 r/m

ST(0) to Long Integer Memory

escape 1 1 1	mod 1 1 1 r/m

ST(0) to Temporary Real Memory

escape 0 1 1	mod 1 1 1 r/m

ST(0) to BCD Memory

escape 1 1 1	mod 1 1 0 r/m

ST(0) to ST(i)

escape 1 0 1	1 1 0 1 1 ST(i)

FXCH = Exchange ST(i) and ST(0)

escape 0 0 1	1 1 0 0 1 ST(i)

Comparison

FCOM = Compare

Integer/Real Memory to ST(0)

escape MF 0	mod 0 1 0 r/m

ST(i) to ST(0)

escape 0 0 0	1 1 0 1 0 ST(i)

FCOMP = Compare and Pop

Integer/Real Memory to ST(0)

escape MF 0	mod 0 1 1 r/m

ST(i) to ST(0)

escape 0 0 0	1 1 0 1 1 ST(i)

FCOMPP = Compare ST(1) to ST(0) and Pop Twice

escape 1 1 0	1 1 0 1 1 0 0 1

FTST = Test ST(0)

escape 0 0 1	1 1 1 0 0 1 0 0

FXAM = Examine ST(0)

escape 0 0 1	1 1 1 0 0 1 0 1

Constants

FLDZ = Load + 0.0 into ST(0)

escape 0 0 1	1 1 1 0 1 1 1 0

FLD1 = Load + 1.0 into ST(0)

escape 0 0 1	1 1 1 0 1 0 0 0

FLDPI = Load π into ST(0)

escape 0 0 1	1 1 1 0 1 0 1 1

FLDL2T = Load $\log_2 10$ into ST(0)

escape 0 0 1	1 1 1 0 1 0 0 1

FLDL2E = Load $\log_2 e$ into ST(0)

escape 0 0 1	1 1 1 0 1 0 1 0

FLDLG2 = Load $\log_{10} 2$ into ST(0)

escape 0 0 1	1 1 1 0 1 1 0 0

FLDLN2 = Load $\log_e 2$ into ST(0)

escape 0 0 1	1 1 1 0 1 1 0 1

Arithmetic

FADD = Addition

Integer/Real Memory with ST(0)

escape MF 0	mod 0 0 0 r/m

ST(i) and ST(0)

escape dP0	1 1 0 0 0 ST(i)

FSUB = Subtraction

Integer/Real Memory with ST(0)

escape MF 0	mod 1 0 R r/m

ST(i) and ST(0)

escape dP 0	1110R r/m

FMUL = Multiplication

Integer/Real Memory with ST(0)

escape MF 0	mod 0 0 1 r/m

ST(i) and ST(0)

escape dP 0	1 1 0 0 1 r/m

FDIV = Division

Integer/Real Memory with ST(0)

escape MF 0	mod 1 1 R r/m

ST(i) and ST(0)

escape dP 0	1 1 1 1 R r/m

FSQRT = Square Root of ST(0)

escape 0 0 1	1 1 1 1 1 0 1 0

FSCALE = Scale ST(0) by ST(1)

escape 0 0 1	1 1 1 1 1 1 0 1

FPREM = Partial Remainder of ST(0) ÷ ST(1)

escape 0 0 1	1 1 1 1 1 0 0 0

FRNDINT = Round ST(0) to Integer

escape 0 0 1	1 1 1 1 1 1 0 0

FXTRACT = Extract Components of ST(0)

escape 0 0 1	1 1 1 1 0 1 0 0

FABS = Absolute Value of ST(0)

escape 0 0 1	1 1 1 0 0 0 0 1

FCHS = Change Sign of ST(0)

escape 0 0 1	1 1 1 0 0 0 0 0

Transcendental

FPTAN = Partial Tangent of ST(0)

escape 0 0 1	1 1 1 1 0 0 1 0

FPATAN = Partial Arctangent of ST(1) ÷ ST(0)

escape 0 0 1	1 1 1 1 0 0 1 1

F2XM1 = $2^{ST(0)}$ -1

escape 0 0 1	1 1 1 1 0 0 0 0

FYL2X = ST(1) x Log_2 [ST(0)]

escape 0 0 1	1 1 1 1 0 0 0 1

FYL2XP1 = ST(1) x Log_2 [ST(0) + 1]

escape 0 0 1	1 1 1 1 1 0 0 1

Processor Control

FINIT = Initialize NPX

escape 0 1 1	1 1 1 0 0 0 1 1

FSETPM = Enter Protected Mode

escape 0 1 1	1 1 1 0 0 1 0 0

FSTSW AX = Store Control Word

escape 1 1 1	1 1 1 0 0 0 0 0

FLDCW = Load Control Word

escape 0 0 1	mod 1 0 1 r/m

FSTCW = Store Control Word

escape 0 0 1	mod 1 1 1 r/m

FSTSW = Store Status Word

escape 1 0 1	mod 1 1 1 r/m

FCLEX = Clear Exceptions

escape 0 1 1	1 1 1 0 0 0 1 0

FSTENV = Store Environment

escape 0 0 1	mod 1 1 0 r/m

FLDENV = Load Environment

escape 0 0 1	mod 1 0 0 r/m

FSAVE = Save State

escape 1 0 1	mod 1 1 0 r/m

FRSTOR = Restore State

escape 1 0 1	mod 1 0 0 r/m

FINCSTP = Increment Stack Pointer

escape 0 0 1	1 1 1 1 0 1 1 1

FDECSTP = Decrement Stack Pointer

escape 0 0 1	1 1 1 1 0 1 1 0

FFREE = Free ST(i)

escape 1 0 1	1 1 0 0 0 ST(i)

FNOP = No Operation

escape 0 0 1	1 1 0 1 0 0 0 0

Introduction to the 80386 Instruction Set

The 80386 instruction set is an extended version of the 8086 and 80286 instruction sets. The instruction sets have been extended in two ways:

- The instructions have extensions that allow operations on 32-bit operands, registers, and memory.
- A 32-bit addressing mode allows flexible selection of registers for base and index as well as index scaling capabilities (x2, x4, x8) for computing a 32-bit effective address. The 32-bit effective address yields a 4GB address range.

Note: The effective address size must be less than 64KB in the real-address or virtual-address modes to avoid an exception.

Code and Data Segment Descriptors

Although the 80386 supports all 80286 Code and Data segment descriptors, there are some differences in the format. The 80286 segment descriptors contain a 24-bit base address and a 16-bit limit field, while the 80386 segment descriptors have a 32-bit base address, a 20-bit limit field, a default bit, and a granularity bit.

31 24	23				16	15 08	07 00	Offset
Segment Base (SB) Bits 15-0						Segment Limit (SL) Bits 15-0		0
SB Bits 31-24	G	D	0	0	SL 19-16	Access Rights Byte	SB Bits 23-16	4

Figure 12. 80386 Code and Data Segment Descriptor Format

Note: Bits 31 through 16 shown at offset 4 are set to 0 for all 80286 segment descriptors.

The default (D) bit of the code segment register is used to determine whether the instruction is carried out as a 16-bit or 32-bit instruction. Code segment descriptors are not used in either the real-address mode or the virtual-8086 mode. When the system microprocessor is operating in either of these modes, a D-bit value of 0 is assumed and operations default to a 16-bit length compatible with 8086 and 80286 programs.

The granularity (G) bit is used to determine the granularity of the segment length (1 = page granular, 0 = byte granular). If the value of the 20 segment-limit bits is defined as *N*, a G-bit value of 1 defines the segment size as follows:

Segment size = $(N + 1) \times 4\text{KB}$

4KB represents the size of a page.

Prefixes

Two prefixes have been added to the instruction set. The Operand Size prefix overrides the default selection of the operand size; the Effective Address Size prefix overrides the effective address size. The presence of either prefix toggles the default setting to its opposite condition. For example:

- If the operand size defaults to 32-bit data operations, the presence of the Operand Size prefix sets it for 16-bit data operations.
- If the effective address size is 16-bits, the presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

The prefixes are available in all 80386 modes, including the real-address mode and the virtual-8086 mode. Since the default of these modes is always 16 bits, the prefixes are used to specify 32-bit operations. If needed, either or both of the prefixes may precede any opcode bytes and affect only the instruction they precede.

Instruction Format

The instructions are presented in this format:

Opcode	Mode Specifier	Address Displacement	Immediate Data

Term	Description
Opcode	The opcode may be one or two bytes in length. Within each byte, smaller encoding fields may be defined.
Mode Specifier	Consists of the "mod r/m" byte and the "scale-index-base" (s-i-b) byte. The mod r/m byte specifies the address mode to be used. Format: mod T T T r/m The "s-i-b" byte is optional and can be used only in 32-bit address modes. It follows the mod r/m byte to fully specify the manner in which the effective address is computed. Format: ss index base
Address Displacement	Follows the "mod r/m" byte or "s-i-b" byte. It may be 8, 16, or 32 bits.
Immediate Data	If specified, follows any displacement bytes and becomes the last field of the instruction. It may be 8, 16, or 32 bits. The term "8-bit data" indicates a fixed data length of 8 bits. The term "8-, 16-, or 32-bit data" indicates a variable data length. The length is determined by the w field and the current operand size. If w = 0, the data is always 8 bits. If w = 1, the size is determined by the operand size of the instruction.

Figure 13. Instruction Format

The instructions use a variety of fields to indicate register selection, the addressing mode, and so on. The following figure is a summary of the fields.

Field Name	Description	Bit Information
w	Specifies if data is byte or full size. (Full size is either 16 or 32 bits.)	1
d	Specifies the direction of data operation.	1
s	Specifies if an immediate data field must be sign-extended.	1
reg	General address specifier.	3
mod r/m	Address mode specifier (effective address can be a general register).	2 for mod; 3 for r/m
ss	Scale factor for scaled index address mode.	2
index	General register to be used as an index register.	3
base	General register to be used as base register.	3
sreg2	Segment register specifier for CS, SS, DS, and ES.	2
sreg3	Segment register specifier for CS, SS, DS, ES, FS, and GS.	3
tttn	For conditional instructions; specifies a condition asserted or a condition negated.	4

Figure 14. 80386 Instruction Set Encoding Field Summary

Encoding

This section defines the encoding of the fields used in the instruction sets.

Address Mode

The first addressing byte is the "mod r/m" byte. The effective address (EA) of the memory operand is computed according to the mod and r/m fields. The mod r/m byte can be interpreted as either a 16-bit or 32-bit addressing mode specifier. Interpretation of the byte depends on the address components used to calculate the EA. The following figure defines the encoding of 16-bit and 32-bit addressing modes with the mod r/m byte.

mod r/m	16-Bit Mode	32-Bit Mode (No s-i-b byte)
00 000	DS:[BX + SI]	DS:[EAX]
00 001	DS:[BX + DI]	DS:[ECX]
00 010	SS:[BP + SI]	DS:[EDX]
00 011	SS:[BP + DI]	DS:[EBX]
00 100	DS:[SI]	s-i-b present (see Figure 19 on page 53)
00 101	DS:[DI]	DS:d32
00 110	d16	DS:[ESI]
00 111	DS:[BX]	DS:[EDI]
01 000	DS:[BX + SI + d8]	DS:[EAX + d8]
01 001	DS:[BX + DI + d8]	DS:[ECX + d8]
01 010	SS:[BP + SI + d8]	DS:[EDX + d8]
01 011	SS:[BP + DI + d8]	DS:[EBX + d8]
01 100	DS:[SI + d8]	s-i-b present (see Figure 19 on page 53)
01 101	DS:[DI + d8]	SS:[EBP + d8]
01 110	SS:[BP + d8]	DS:[ESI + d8]
01 111	DS:[BX + d8]	DS:[EDI + d8]
10 000	DS:[BX + SI + d16]	DS:[EAX + d32]
10 001	DS:[BX + DI + d16]	DS:[ECX + d32]
10 010	SS:[BP + SI + d16]	SS:[EDX + d32]
10 011	SS:[BP + DI + d16]	DS:[EBX + d32]
10 100	DS:[SI + d16]	s-i-b present (see Figure 19 on page 53)
10 101	DS:[DI + d16]	SS:[EBP + d32]
10 110	SS:[BP + d16]	DS:[ESI + d32]
10 111	DS:[BX + d16]	DS:[EDI + d32]

Figure 15. Effective Address (16-Bit and 32-Bit Address Modes)

The displacement follows the second byte of the instruction (before data, if required).

The scale-index-base (s-i-b) byte can be specified as a second byte of addressing information. The s-i-b byte is specified when using a 32-bit addressing mode and the mod r/m byte has the following values:

- r/m = 100
- mod = 00, 01, or 10.

When the s-i-b byte is present, the 32-bit effective address is a function of the mod, ss, index, and base fields. The following figures show the scale factor, index register selected, and base register selected when the s-i-b byte is present.

ss	**Scale Factor**
00	1
01	2
10	4
11	8

Figure 16. Scale Factor (s-i-b Byte Present)

Index	**Index Register**	
000	EAX	
001	ECX	
010	EDX	
011	EBX	
100	No Index Register	The ss field must equal 00 when the index field is 100; if not, the effective address is undefined.
101	EBP	
110	ESI	
111	EDI	

Figure 17. Index Registers (s-i-b Byte Present)

base	Base Register	
000	EAX	
001	ECX	
010	EDX	
011	EBX	
100	ESP	
101	EBP	If mod = 00, then EBP is not used to form the EA; immediate 32-bit address displacement follows the mode specifier byte.
110	ESI	
111	EDI	

Figure 18. Base Registers (s-i-b Byte Present)

The scaled-index information is determined by multiplying the contents of the Index register by the scale factor. The following example shows the use of the 32-bit addressing mode with scaling where:

- EAX is the base of ARRAY_A
- ECX is the index of the desired element
- 2 is the scale factor.

```
; ARRAY_A is an array of words
MOV EAX, offset ARRAY_A
MOV ECX, element_number
MOV BX, [EAX][ECX*2]
```

The following figure defines the encoding of the 32-bit addressing mode when the s-i-b byte is present.

Note: The mod field is from the mod r/m byte. The base field and scaled-index information are from the s-i-b byte.

Mod Base	32-Bit Address Mode
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

Figure 19. Effective Address (32-Bit Address Mode — s-i-b Byte Present)

Operand Length (w) Field

For an instruction performing a data operation, the instruction is executed as either a 32-bit or 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or full operation.

w	16-Bit Data Operation	32-Bit Data Operation
0	8 Bits	8 Bits
1	16 Bits	32 Bits

Figure 20. Operand Length Field Encoding

Segment Register (sreg) Field

The 2-bit segment register field (sreg2) allows one of the four 80286 segment registers to be specified. The 3-bit segment register (sreg3) allows the 80386 FS and GS segment registers to be specified.

sreg2	sreg3	Segment Register
00	000	ES
01	001	CS
10	010	SS
11	011	DS
--	100	FS
--	101	GS
--	110	Reserved
--	111	Reserved

Figure 21. Segment Register Field Encoding

General Register (reg) Field

The general register is specified by the reg field, which may appear in the primary opcode bytes as the reg field of the mod reg r/m byte, or as the r/m field of the mod reg r/m byte when mod = 11.

reg	16-Bit w/o w	16-Bit w=0	16-Bit w=1	32-Bit w/o w	32-Bit w=0	32-Bit w=1
000	AX	AL	AX	EAX	AL	EAX
001	CX	CL	CX	ECX	CL	ECX
010	DX	DL	DX	EDX	DL	EDX
011	BX	BL	BX	EBX	BL	EBX
100	SP	AH	SP	ESP	AH	ESP
101	BP	CH	BP	EBP	CH	EBP
110	SI	DH	SI	ESI	DH	ESI
111	DI	BH	DI	EDI	BH	EDI

Figure 22. General Register Field Encoding

The physical addresses of all operands addressed by the BP register are computed using the SS Segment register. For string primitive operations (those addressed by the DI register), addresses of the destination operands are computed using the ES segment, which may not be overridden.

Operation Direction (d) Field

The operation direction (d) field is used in many two-operand instructions to indicate which operand is the source and which is the destination.

d	Direction of Operation
0	Register/Memory <-- Register The "reg" field indicates the source operand; "mod r/m" or "mod ss index base" indicates the destination operand.
1	Register<-- Register/Memory The "reg" field indicates the destination operand; "mod r/m" or "mod ss index base" indicates the source operand.

Figure 23. Operand Direction Field Encoding

Sign-Extend (s) Field

The sign-extend (s) field appears primarily in instructions having immediate data fields. The s field affects only 8-bit immediate data being placed in a 16-bit or 32-bit destination.

s	8-Bit Immediate Data	16/32-Bit Immediate Data
0	No effect on data	No effect on data
1	Sign-extend 8-bit data to fill 16-bit or 32-bit destination	No effect on data

Figure 24. Sign-Extend Field Encoding

Conditional Test (tttn) Field

For conditional instructions (conditional jumps and set-on condition), the conditional test (tttn) field is encoded, with n indicating whether to use the condition (n = 0) or its negation (n = 1), and ttt defining the condition to test.

tttn	Condition	Mnemonic
0000	Overflow	O
0001	No Overflow	NO
0010	Below/Not Above or Equal	B/NAE
0011	Not Below/Above or Equal	NB/AE
0100	Equal/Zero	E/Z
0101	Not Equal/Not Zero	NE/NZ
0110	Below or Equal/Not Above	BE/NA
0111	Not Below or Equal/Above	NBE/A
1000	Sign	S
1001	Not Sign	NS
1010	Parity/Parity Even	P/PE
1011	Not Parity/Parity Odd	NP/PO
1100	Less Than/Not Greater or Equal	L/NGE
1101	Not Less Than/Greater or Equal	NL/GE
1110	Less Than or Equal/Not Greater Than	LE/NG
1111	Not Less or Equal/Greater Than	NLE/G

Figure 25. Conditional Test Field Encoding

Control, Debug, or Test Register (eee) Field

The following shows the encoding for loading and storing the Control, Debug, and Test registers (eee).

eee Code	Interpreted as Control Register	Interpreted as Debug Register	Interpreted as Test Register
000	CR0	DR0	---
001	---	DR1	---
010	CR2	DR2	---
011	CR3	DR3	---
100	---	---	---
101	---	---	---
110	---	DR6	TR6
111	---	DR7	TR7

Figure 26. Control, Debug, and Test Register Field Encoding

80386 Microprocessor Instruction Set

Data Transfer

MOV = Move

Register to Register/Memory

1 0 0 0 1 0 0 w	mod reg r/m

Register/Memory to Register

1 0 0 0 1 0 1 w	mod reg r/m

Immediate to Register/Memory

1 1 0 0 0 1 1 w	mod 0 0 0 r/m	8-, 16-, or 32-bit data

Immediate to Register (Short Form)

1 0 1 1 w reg	8-, 16-, or 32-bit data

Memory to Accumulator (Short Form)

1 0 1 0 0 0 0 w	full 16- or 32-bit displacement

Accumulator to Memory (Short Form)

1 0 1 0 0 0 1 w	full 16- or 32-bit displacement

Register/Memory to Segment Register

1 0 0 0 1 1 1 0	mod sreg3 r/m

Segment Register to Register/Memory

1 0 0 0 1 1 0 0	mod sreg3 r/m

MOVSX = Move with Sign Extension

Register from Register/Memory

00001111	1011111w	mod reg r/m

MOVZX = Move with Zero Extension

Register from Register/Memory

00001111	1011011w	mod reg r/m

PUSH = Push

Register/Memory

11111111	mod 110 r/m

Register (Short Form)

01010 reg

Segment Register (ES, CS, SS, or DS) Short Form

000 sreg2 110

Segment Register (FS or GS)

00001111	10 sreg3 000

Immediate

011010s0	8-, 16-, or 32-bit data

PUSHA = Push All

01100000

POP = Pop

Register/Memory

1 0 0 0 1 1 1 1	mod 0 0 0 r/m

Register (Short Form)

0 1 0 1 1 reg

Segment Register (ES, SS, or DS) Short Form

0 0 0 sreg2 1 1 1

Segment Register (FS or GS)

0 0 0 0 1 1 1 1	1 0 sreg3 0 0 1

POPA = Pop All

0 1 1 0 0 0 0 1

XCHG = Exchange

Register/Memory with Register

1 0 0 0 0 1 1 w	mod reg r/m

Register with Accumulator (Short Form)

1 0 0 1 0 reg

IN = Input From:

Fixed Port

1 1 1 0 0 1 0 w	port number

Variable Port

1 1 1 0 1 1 0 w

OUT = Output To:

Fixed Port

1 1 1 0 0 1 1 w	port number

Variable Port

1 1 1 0 1 1 1 w

LEA = Load EA to Register

1 0 0 0 1 1 0 1	mod reg r/m

Segment Control

LDS = Load Pointer to DS

1 1 0 0 0 1 0 1	mod reg r/m

LES = Load Pointer to ES

1 1 0 0 0 1 0 0	mod reg r/m

LFS = Load Pointer to FS

0 0 0 0 1 1 1 1	1 0 1 1 0 1 0 0	mod reg r/m

LGS = Load Pointer to GS

0 0 0 0 1 1 1 1	1 0 1 1 0 1 0 1	mod reg r/m

LSS = Load Pointer to SS

0 0 0 0 1 1 1 1	1 0 1 1 0 0 1 0	mod reg r/m

Flag Control

CLC = Clear Carry Flag

11111000

CLD = Clear Direction Flag

11111100

CLI = Clear Interrupt Enable Flag

11111010

CLTS = Clear Task Switched Flag

00001111	00000110

CMC = Complement Carry Flag

11110101

LAHF = Load AH Into Flag

10011111

POPF = Pop Flags

10011101

PUSHF = Push Flags

10011100

SAHF = Store AH into Flags

1 0 0 1 1 1 1 0

STC = Set Carry Flag

1 1 1 1 1 0 0 1

STD = Set Direction Flag

1 1 1 1 1 1 0 1

STI = Set Interrupt Enable Flag

1 1 1 1 1 0 1 1

Arithmetic

ADD = Add

Register to Register

0 0 0 0 0 0 d w	mod reg r/m

Register to Memory

0 0 0 0 0 0 0 w	mod reg r/m

Memory to Register

0 0 0 0 0 0 1 w	mod reg r/m

Immediate to Register/Memory

1 0 0 0 0 0 s w	mod 0 0 0 r/m	8-, 16-, or 32-bit data

Immediate to Accumulator (Short Form)

0 0 0 0 0 1 0 w	8-, 16-, or 32-bit data

ADC = Add with Carry

Register to Register

0 0 0 1 0 0 d w	mod reg r/m

Register to Memory

0 0 0 1 0 0 0 w	mod reg r/m

Memory to Register

0 0 0 1 0 0 1 w	mod reg r/m

Immediate to Register/Memory

1 0 0 0 0 0 s w	mod 0 1 0 r/m	8-, 16-, or 32-bit data

Immediate to Accumulator (Short Form)

0 0 0 1 0 1 0 w	8-, 16-, or 32-bit data

INC = Increment

Register/Memory

1 1 1 1 1 1 1 w	mod 0 0 0 r/m

Register (Short Form)

0 1 0 0 0 reg

SUB = Subtract

Register from Register

0 0 1 0 1 0 d w	mod reg r/m

Register from Memory

0 0 1 0 1 0 0 w	mod reg r/m

Memory from Register

0 0 1 0 1 0 1 w	mod reg r/m

Immediate from Register/Memory

1 0 0 0 0 0 s w	mod 1 0 1 r/m	8-, 16-, or 32-bit data

Immediate from Accumulator (Short Form)

0 0 1 0 1 1 0 w	8-, 16-, or 32-bit data

SBB = Subtract with Borrow

Register from Register

0 0 0 1 1 0 d w	mod reg r/m

Register from Memory

0 0 0 1 1 0 0 w	mod reg r/m

Memory from Register

0 0 0 1 1 0 1 w	mod reg r/m

Immediate from Register/Memory

1 0 0 0 0 0 s w	mod 0 1 1 r/m	8-, 16-, or 32-bit data

Immediate from Accumulator (Short Form)

0 0 0 1 1 1 0 w	8-, 16-, or 32-bit data

DEC = Decrement

Register/Memory

1 1 1 1 1 1 1 w	mod 0 0 1 r/m

Register (Short Form)

0 1 0 0 1 reg

CMP = Compare

Register with Register

0 0 1 1 1 0 d w	mod reg r/m

Memory with Register

0 0 1 1 1 0 0 w	mod reg r/m

Register with Memory

0 0 1 1 1 0 1 w	mod reg r/m

Immediate with Register/Memory

1 0 0 0 0 0 s w	mod 1 1 1 r/m	8-, 16-, or 32-bit data

Immediate with Accumulator (Short Form)

0 0 1 1 1 1 0 w	8-, 16-, or 32-bit data

NEG = Change Sign

1 1 1 1 0 1 1 w	mod 0 1 1 r/m

AAA = ASCII Adjust for Add

0 0 1 1 0 1 1 1

AAS = ASCII Adjust for Subtract

0 0 1 1 1 1 1 1

DAA = Decimal Adjust for Add

0 0 1 0 0 1 1 1

DAS = Decimal Adjust for Subtract

0 0 1 0 1 1 1 1

MUL = Multiply (Unsigned)

Accumulator with Register/Memory

1 1 1 1 0 1 1 w	mod 1 0 0 r/m

IMUL = Integer Multiply (Signed)

Accumulator with Register/Memory

1 1 1 1 0 1 1 w	mod 1 0 1 r/m

Register with Register/Memory

0 0 0 0 1 1 1 1	1 0 1 0 1 1 1 1	mod reg r/m

Register/Memory with Immediate to Register

0 1 1 0 1 0 s 1	mod reg r/m	8-, 16-, or 32-bit data

DIV = Divide (Unsigned)

Accumulator by Register/Memory

1 1 1 1 0 1 1 w	mod 1 1 0 r/m

IDIV = Integer Divide (Signed)

Accumulator by Register/Memory

1 1 1 1 0 1 1 w	mod 1 1 1 r/m

AAD = ASCII Adjust for Divide

1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0

AAM = ASCII Adjust for Multiply

1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0

CBW = Convert Byte to Word

1 0 0 1 1 0 0 0

CWD = Convert Word to Doubleword

1 0 0 1 1 0 0 1

Logic

Shift/Rotate Instructions
Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)

Register/Memory by 1

1 1 0 1 0 0 0 w	mod T T T r/m

Register/Memory by CL

1 1 0 1 0 0 1 w	mod T T T r/m

Register/Memory by Immediate Count

1 1 0 0 0 0 0 w	mod T T T r/m	8-bit data

Shift/Rotate Instructions Through Carry (RCL and RCR)

Register/Memory by 1

1 1 0 1 0 0 0 w	mod T T T r/m

Register/Memory by CL

1 1 0 1 0 0 1 w	mod T T T r/m

Register/Memory by Immediate Count

1 1 0 0 0 0 0 w	mod T T T r/m	8-bit data

T T T	Instruction
0 0 0	ROL
0 0 1	ROR
0 1 0	RCL
0 1 1	RCR
1 0 0	SHL/SAL
1 0 1	SHR
1 1 1	SAR

SHLD = Shift Left Double

Register/Memory by Immediate

0 0 0 0 1 1 1 1	1 0 1 0 0 1 0 0	mod reg r/m	8-bit data

Register/Memory by CL

0 0 0 0 1 1 1 1	1 0 1 0 0 1 0 1	mod reg r/m

SHRD = Shift Right Double

Register/Memory by Immediate

0 0 0 0 1 1 1 1	1 0 1 0 1 1 0 0	mod reg r/m	8-bit data

Register/Memory by CL

0 0 0 0 1 1 1 1	1 0 1 0 1 1 0 1	mod reg r/m

AND = And

Register to Register

0 0 1 0 0 0 d w	mod reg r/m

Register to Memory

0 0 1 0 0 0 0 w	mod reg r/m

Memory to Register

0 0 1 0 0 0 1 w	mod reg r/m

Immediate to Register/Memory

1 0 0 0 0 0 s w	mod 1 0 0 r/m	8-, 16-, or 32-bit data

Immediate to Accumulator (Short Form)

0 0 1 0 0 1 0 w	8-, 16-, or 32-bit data

TEST = AND Function to Flags; No Result

Register/Memory and Register

1 0 0 0 0 1 0 w	mod reg r/m

Immediate Data and Register/Memory

1 1 1 1 0 1 1 w	mod 0 0 0 r/m	8-, 16-, or 32-bit data

Immediate Data and Accumulator (Short Form)

1 0 1 0 1 0 0 w	8-, 16-, or 32-bit data

OR = Or

Register to Register

0 0 0 0 1 0 d w	mod reg r/m

Register to Memory

0 0 0 0 1 0 0 w	mod reg r/m

Memory to Register

0 0 0 0 1 0 1 w	mod reg r/m

Immediate to Register/Memory

1 0 0 0 0 0 s w	mod 0 0 1 r/m	8-, 16-, or 32-bit data

Immediate to Accumulator (Short Form)

0 0 0 0 1 1 0 w	8-, 16-, or 32-bit data

XOR = Exclusive OR

Register to Register

0 0 1 1 0 0 d w	mod reg r/m

Register to Memory

0 0 1 1 0 0 0 w	mod reg r/m

Memory to Register

0 0 1 1 0 0 1 w	mod reg r/m

Immediate to Register/Memory

1 0 0 0 0 0 s w	mod 1 1 0 r/m	8-, 16-, or 32-bit data

Immediate to Accumulator (Short Form)

0 0 1 1 0 1 0 w	8-, 16-, or 32-bit data

NOT = Invert Register/Memory

1 1 1 1 0 1 1 w	mod 0 1 0 r/m

String Manipulation

CMPS = Compare Byte Word

1 0 1 0 0 1 1 w

INS = Input Byte/Word from DX Port

0 1 1 0 1 1 0 w

LODS = Load Byte/Word to AL/AX/EAX

1 0 1 0 1 1 0 w

MOVS = Move Byte Word

1 0 1 0 0 1 0 w

OUTS = Output Byte/Word to DX Port

0 1 1 0 1 1 1 w

SCAS = Scan Byte Word

1 0 1 0 1 1 1 w

STOS = Store Byte/Word from AL/AX/EX

1 0 1 0 1 0 1 w

XLAT = Translate String

1 1 0 1 0 1 1 1

Repeated String Manipulation

Repeated by Count in CX or ECX

REPE CMPS = Compare String (Find Non-Match)

1 1 1 1 0 0 1 1	1 0 1 0 0 1 1 w

REPNE CMPS = Compare String (Find Match)

11110010	1010011w

REP INS = Input String

11110010	0110110w

REP LODS = Load String

11110010	1010110w

REP MOVS = Move String

11110010	1010010w

REP OUTS = Output String

11110010	0110111w

REPE SCAS = Scan String (Find Non-AL/AX/EAX)

11110011	1010111w

REPNE SCAS = Scan String (Find AL/AX/EAX)

11110010	1010111w

REP STOS = Store String

11110010	1010101w

Bit Manipulation

BSF = Scan Bit Forward

00001111	10111100	mod reg r/m

BSR = Scan Bit Reverse

00001111	10111101	mod reg r/m

BT = Test Bit

Register/Memory, Immediate

00001111	10111010	mod 1 0 0 r/m	8-bit data

Register/Memory, Register

00001111	10100011	mod reg r/m

BTC = Test Bit and Complement

Register/Memory, Immediate

00001111	10111010	mod 1 1 1 r/m	8-bit data

Register/Memory, Register

00001111	10111011	mod reg r/m

BTR = Test Bit and Reset

Register/Memory, Immediate

00001111	10111010	mod 1 1 0 r/m	8-bit data

Register/Memory, Register

00001111	10110011	mod reg r/m

BTS = Test Bit and Set

Register/Memory, Immediate

0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod 1 0 1 r/m	8-bit data

Register/Memory, Register

0 0 0 0 1 1 1 1	1 0 1 0 1 0 1 1	mod reg r/m

Control Transfer

CALL = Call

Direct within Segment

1 1 1 0 1 0 0 0	full 16- or 32-bit displacement

Register/Memory Indirect within Segment

1 1 1 1 1 1 1 1	mod 0 1 0 r/m

Direct Intersegment

1 0 0 1 1 0 1 0	offset, selector

Indirect Intersegment

1 1 1 1 1 1 1 1	mod 0 1 1 r/m

JMP = Unconditional Jump

Short

1 1 1 0 1 0 1 1	8-bit disp.

Direct within Segment

1 1 1 0 1 0 0 1	full 16- or 32-bit displacement

Register/Memory Indirect within Segment

1 1 1 1 1 1 1 1	mod 1 0 0 r/m

Direct Intersegment

1 1 1 0 1 0 1 0	offset, selector

Indirect Intersegment

1 1 1 1 1 1 1 1	mod 1 0 1 r/m

RET = Return from Call

Within Segment

1 1 0 0 0 0 1 1

Within Segment Adding Immediate to SP

1 1 0 0 0 0 1 0	16-bit displacement

Intersegment

1 1 0 0 1 0 1 1

Intersegment Adding Immediate to SP

1 1 0 0 1 0 1 0	16-bit displacement

Conditional Jumps

JO = Jump on Overflow

8-Bit Displacement

0 1 1 1 0 0 0 0	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 0 0 0 0	full 16- or 32-bit displacement

JNO = Jump on Not Overflow

8-Bit Displacement

0 1 1 1 0 0 0 1	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 0 0 0 1	full 16- or 32-bit displacement

JB/JNAE = Jump on Below/Not Above or Equal

8-Bit Displacement

0 1 1 1 0 0 1 0	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 0 0 1 0	full 16- or 32-bit displacement

JNB/JAE = Jump on Not Below/Above or Equal

8-Bit Displacement

0 1 1 1 0 0 1 1	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 0 0 1 1	full 16- or 32-bit displacement

JE/JZ = Jump on Equal/Zero

8-Bit Displacement

0 1 1 1 0 1 0 0	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 0 1 0 0	full 16- or 32-bit displacement

JNE/JNZ = Jump on Not Equal/Not Zero

8-Bit Displacement

0 1 1 1 0 1 0 1	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 0 1 0 1	full 16- or 32-bit displacement

JBE/JNA = Jump on Below or Equal/Not Above

8-Bit Displacement

0 1 1 1 0 1 1 0	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 0 1 1 0	full 16- or 32-bit displacement

JNBE/JA = Jump on Not Below or Equal/Above

8-Bit Displacement

0 1 1 1 0 1 1 1	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 0 1 1 1	full 16- or 32-bit displacement

JS = Jump on Sign

8-Bit Displacement

0 1 1 1 1 0 0 0	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 1 0 0 0	full 16- or 32-bit displacement

JNS = Jump on Not Sign

8-Bit Displacement

0 1 1 1 1 0 0 1	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 1 0 0 1	full 16- or 32-bit displacement

JP/JPE = Jump on Parity/Parity Even

8-Bit Displacement

0 1 1 1 1 0 1 0	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 1 0 1 0	full 16- or 32-bit displacement

JNP/JPO = Jump on Not Parity/Parity Odd

8-Bit Displacement

0 1 1 1 1 0 1 1	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 1 0 1 1	full 16- or 32-bit displacement

JL/JNGE = Jump on Less/Not Greater or Equal

8-Bit Displacement

0 1 1 1 1 1 0 0	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 1 1 0 0	full 16- or 32-bit displacement

JNL/JGE = Jump on Not Less/Greater or Equal

8-Bit Displacement

0 1 1 1 1 1 0 1	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 1 1 0 1	full 16- or 32-bit displacement

JLE/JNG = Jump on Less or Equal/Not Greater

8-Bit Displacement

0 1 1 1 1 1 1 0	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 1 1 1 0	full 16- or 32-bit displacement

JNLE/JG = Jump on Not Less or Equal/Greater

8-Bit Displacement

0 1 1 1 1 1 1 1	8-bit disp.

Full Displacement

0 0 0 0 1 1 1 1	1 0 0 0 1 1 1 1	full 16- or 32-bit displacement

JCXZ = Jump on CX Zero

1 1 1 0 0 0 1 1	8-bit disp.

JECXZ = Jump on ECX Zero

1 1 1 0 0 0 1 1	8-bit disp.

Note: The operand size prefix differentiates JCXZ from JECXZ.

LOOP = Loop CX Times

11100010	8-bit disp.

LOOPZ/LOOPE = Loop with Zero/Equal

11100001	8-bit disp.

LOOPNZ/LOOPNE = Loop while Not Zero

11100000	8-bit disp.

Conditional Byte Set

SETO = Set Byte on Overflow

To Register/Memory

00001111	10010000	mod 000 r/m

SETNO = Set Byte on Not Overflow

To Register/Memory

00001111	10010001	mod 000 r/m

SETB/SETNAE = Set Byte on Below/Not Above or Equal

To Register/Memory

00001111	10010010	mod 000 r/m

SETNB = Set Byte on Not Below/Above or Equal

To Register/Memory

00001111	10010011	mod 000 r/m

SETE/SETZ = Set Byte on Equal/Zero

To Register/Memory

00001111	10010100	mod 000 r/m

SETNE/SETNZ = Set Byte on Not Equal/Not Zero

To Register/Memory

0 0 0 0 1 1 1 1	1 0 0 1 0 1 0 1	mod 0 0 0 r/m

SETBE/SETNA = Set Byte on Below or Equal/Not Above

To Register/Memory

0 0 0 0 1 1 1 1	1 0 0 1 0 1 1 0	mod 0 0 0 r/m

SETNBE/SETA = Set Byte on Not Below or Equal/Above

To Register/Memory

0 0 0 0 1 1 1 1	1 0 0 1 0 1 1 1	mod 0 0 0 r/m

SETS = Set Byte on Sign

To Register/Memory

0 0 0 0 1 1 1 1	1 0 0 1 1 0 0 0	mod 0 0 0 r/m

SETNS = Set Byte on Not Sign

To Register/Memory

0 0 0 0 1 1 1 1	1 0 0 1 1 0 0 1	mod 0 0 0 r/m

SETP/SETPE = Set Byte on Parity/Parity Even

To Register/Memory

0 0 0 0 1 1 1 1	1 0 0 1 1 0 1 0	mod 0 0 0 r/m

SETNP/SETPO = Set Byte on Not Parity/Parity Odd

To Register/Memory

0 0 0 0 1 1 1 1	1 0 0 1 1 0 1 1	mod 0 0 0 r/m

SETL/SETNGE = Set Byte on Less/Not Greater or Equal

To Register/Memory

0 0 0 0 1 1 1 1	1 0 0 1 1 1 0 0	mod 0 0 0 r/m

SETNL/SETGE = Set Byte on Not Less/Greater or Equal

To Register/Memory

0 0 0 0 1 1 1 1	0 1 1 1 1 1 0 1	mod 0 0 0 r/m

SETLE/SETNG = Set Byte on Less or Equal/Not Greater

To Register/Memory

0 0 0 0 1 1 1 1	1 0 0 1 1 1 1 0	mod 0 0 0 r/m

SETNLE/SETG = Set Byte on Not Less or Equal/Greater

To Register/Memory

0 0 0 0 1 1 1 1	1 0 0 1 1 1 1 1	mod 0 0 0 r/m

ENTER = Enter Procedure

1 1 0 0 1 0 0 0	16-bit displacement	8-bit level

LEAVE = Leave Procedure

1 1 0 0 1 0 0 1

Interrupt Instructions

INT = Interrupt

Type Specified

1 1 0 0 1 1 0 1	type

Type 3

1 1 0 0 1 1 0 0

INTO = Interrupt 4 if Overflow Flag Set

1 1 0 0 1 1 1 0

BOUND = Interrupt 5 if Detect Value Out of Range

01100010	mod reg r/m

IRET = Interrupt Return

11001111

Processor Control

HLT = Halt

11110100

MOV = Move to and from Control/Debug/Test Registers

CR0/CR2/CR3 from Register

00001111	00100010	1 1 eee reg

Register from CR0-3

00001111	00100000	1 1 eee reg

DR0-3, DR6-7 from Register

00001111	00100011	1 1 eee reg

Register from DR0-3, DR6-7

00001111	00100001	1 1 eee reg

TR6-7 from Register

00001111	00100110	1 1 eee reg

Register from TR6-7

00001111	00100100	1 1 eee reg

NOP = No Operation

10010000

WAIT = Wait until BUSY Pin is Negated

1 0 0 1 1 0 1 1

Processor Extension

ESC = Processor Extension Escape

1 1 0 1 1 T T T	mod L L L r/m

Note: TTT and LLL bits are opcode information for the coprocessor.

Prefix Bytes

Address Size Prefix

0 1 1 0 0 1 1 1

Operand Size Prefix

0 1 1 0 0 1 1 0

LOCK = Bus Lock Prefix

1 1 1 1 0 0 0 0

Note: The use of LOCK is restricted to an exchange with memory, or bit test and reset type of instruction.

Segment Override Prefix

CS:

0 0 1 0 1 1 1 0

DS:

0 0 1 1 1 1 1 0

ES:

0 0 1 0 0 1 1 0

FS:

0 1 1 0 0 1 0 0

GS:

0 1 1 0 0 1 0 1

SS:

0 0 1 1 0 1 1 0

Protection Control

ARPL = Adjust Requested Privilege Level from Register/Memory

0 1 1 0 0 0 1 1	mod reg r/m

LAR = Load Access Rights from Register/Memory

0 0 0 0 1 1 1 1	0 0 0 0 0 0 1 0	mod reg r/m

LGDT = Load Global Descriptor Table Register

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 1 0 r/m

LIDT = Load Interrupt Descriptor Table Register

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 1 1 r/m

LLDT = Load Local Descriptor Table Register to Register/Memory

0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 1 0 r/m

.MSW = Load Machine Status Word from Register/Memory

00001111	00000001	mod 1 1 0 r/m

LSL = Load Segment Limit from Register/Memory

00001111	00000011	mod reg r/m

LTR = Load Task Register from Register/Memory

00001111	00000000	mod 0 0 1 r/m

SGDT = Store Global Descriptor Table Register

00001111	00000001	mod 0 0 0 r/m

SIDT = Store Interrupt Descriptor Table Register

00001111	00000001	mod 0 0 1 r/m

SLDT = Store Local Descriptor Table Register to Register/Memory

00001111	00000000	mod 0 0 0 r/m

SMSW = Store Machine Status Word

00001111	00000001	mod 1 0 0 r/m

STR = Store Task Register to Register/Memory

00001111	00000000	mod 0 0 1 r/m

VERR = Verify Read Access; Register/Memory

00001111	00000000	mod 1 0 0 r/m

VERW = Verify Write Access

00001111	00000000	mod 1 0 1 r/m

Introduction to the 80387 Instruction Set

The 80387 instructions use many of the same fields defined earlier in this section for the 80386 instructions. Additional fields used by the 80387 instructions are defined in the following figure.

Field	Description	Bit Information
escape	80386 Extension Escape	Bit Pattern = 11011
MF	Memory Format	00 = 32-bit Real 01 = 32-bit integer 10 = 64-bit Real 11 = 16-bit integer
ST(0)	Current Stack Top	
ST(i)	ith register below the stack top	
d	Destination	0 = Destination is ST(0) 1 = Destination is ST(i)
P	Pop	0 = No pop 1 = Pop ST(0)
R	Reverse*	0 = Destination (op) source 1 = Source (op) destination

* When d = 1, reverse the sense of R.

Figure 27. 80387 Encoding Field Summary

Within the 80387 Instruction Set:

- Temporary (Extended) Real is 80-bit Real.
- Long Integer is a 64-bit integer.

80387 Usage of the Scale-Index-Base Byte

The "mod r/m" byte of an 80387 instruction can be followed by a scale-index-base (s-i-b) byte having the same address mode definition as in the 80386 instruction. The mod field in the 80387 instruction is never equal to 11.

Instruction and Data Pointers

The parallel operation of the 80386 and 80387 may allow errors detected by the 80387 to be reported after the 80386 has executed the ESC instruction that caused the error. The 80386/80387 provides two pointer registers to identify the failing numeric instruction. The pointer registers supply the address of the failing numeric instruction and the address of its numeric memory operand when applicable.

Although the pointer registers are located in the 80386, they appear to be located in the 80387 because they are accessed by the ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR. Whenever the 80386 decodes a new ESC instruction, it saves the address of the instruction along with any prefix bytes that may be present, the address of the operand (if present), and the opcode.

The instruction and data pointers appear in one of four available formats:

- 16-bit Real Mode/Virtual 8086 Mode
- 32-bit Real Mode
- 16-bit Protected Mode
- 32-bit Protected Mode

The Real Mode formats are used whenever the 80386 is in the Real Mode or Virtual 8086 Mode. The Protected Mode formats are used when the 80386 is in the Protected Mode. The Operand Size Prefix can also be used with the 80387 instructions. The operand size of the 80387 instruction determines whether the 16-bit or 32-bit format is used.

Note: FSAVE and FRSTOR have an additional eight fields (10 bytes per field) that contain the current contents of ST(0) through ST(7). These fields follow the instruction and data pointer image shown in the following figures.

The following figures show the instruction and data pointer image format used in the various address modes. The ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR are used to transfer these values between the 80386/80387 registers and memory.

Bits

15		8 | 7 0	Offset in Block
Control Word			0
Status Word			2
Tag Word			4
Instruction Pointer (IP) Bits 15-0			6
IP Bits 19-16	0	Opcode Bits 10-0	8
Operand Pointer (OP) Bits 15-0			A
OP Bits 19-16	0	0 0 0 0 0 0 0 0 0 0 0	C

Figure 28. Instruction and Pointer Image (16-Bit Real Address Mode)

Bits

15 8	7 0	Offset in Block
Control Word		0
Status Word		2
Tag Word		4
Instruction Pointer Offset		6
CS Selector		8
Operand Offset		A
Operand Selector		C

Figure 29. Instruction and Pointer Image (16-Bit Protected Mode)

Bits

31 24	23 16	15 8	7 0	Offset in Block
Reserved		Control Word		0
Reserved		Status Word		4
Reserved		Tag Word		8
Reserved		IP Bits 15-0		C
0 0 0 0	IP Bits 31-16	0	Opcode Bits 10-0	10
Reserved		Operand Pointer Bits 15-0		14
0 0 0 0	Operand Pointer Bits 31-16		0 0 0 0 0 0 0 0 0 0 0 0	18

Figure 30. Instruction and Pointer Image (32-Bit Real Address Mode)

Bits

31 24	23 16	15 8	7 0	Offset in Block
Reserved		Control Word		0
Reserved		Status Word		4
Reserved		Tag Word		8
Instruction Pointer Offset				C
Reserved		CS Selector		10
Data Operand Offset				14
Reserved		Operand Selector		18

Figure 31. Instruction and Pointer Image (32-Bit Protected Mode)

New Instructions

Several new instructions are included in the 80387 instruction set that are not available to the 80287 or 8087 Math Coprocessors. The new instructions are:

FUCOM (Unordered Compare Real)
FUCOMP (Unordered Compare Real and Pop)
FUCOMPP (Unordered Compare Real and Pop Twice)
FPREM1 (IEEE Partial Remainder)
FSINE (Sine)
FCOS (Cosine)
FSINCOS (Sine and Cosine).

80387 Math Coprocessor Instruction Set

The following is an instruction set summary for the 80387 coprocessor. In the following, the bit pattern for escape is 11011.

Data Transfer

FLD = Load

Integer/Real Memory to ST(0)

escape MF 1	mod 0 0 0 r/m

Long Integer Memory to ST(0)

escape 1 1 1	mod 1 0 1 r/m

Temporary Real Memory to ST(0)

escape 0 1 1	mod 1 0 1 r/m

BCD Memory to ST(0)

escape 1 1 1	mod 1 0 0 r/m

ST(i) to ST(0)

escape 0 0 1	1 1 0 0 0 ST(i)

FST = Store

ST(0) to Integer/Real Memory

escape MF 1	mod 0 1 0 r/m

ST(0) to ST(i)

escape 1 0 1	1 1 0 1 0 ST(i)

FSTP = Store and Pop

ST(0) to Integer/Real Memory

escape MF 1	mod 0 1 1 r/m

ST(0) to Long Integer Memory

escape 1 1 1	mod 1 1 1 r/m

ST(0) to Temporary Real Memory

escape 0 1 1	mod 1 1 1 r/m

ST(0) to BCD Memory

escape 1 1 1	mod 1 1 0 r/m

ST(0) to ST(i)

escape 1 0 1	1 1 0 1 1 ST(i)

FXCH = Exchange ST(i) and ST(0)

escape 0 0 1	1 1 0 0 1 ST(i)

Comparison

FCOM = Compare

Integer/Real Memory to ST(0)

escape MF 0	mod 0 1 0 r/m

ST(i) to ST(0)

escape 0 0 0	1 1 0 1 0 ST(i)

FCOMP = Compare and Pop

Integer/Real Memory to ST(0)

escape MF 0	mod 0 1 1 r/m

ST(i) to ST(0)

escape 0 0 0	1 1 0 1 1 ST(i)

FCOMPP = Compare ST(1) to ST(0) and Pop Twice

escape 1 1 0	1 1 0 1 1 0 0 1

FUCOM = Unordered Compare Real

escape 1 0 1	1 1 1 0 0 ST(i)

FUCOMP = Unordered Compare Real and Pop

escape 1 0 1	1 1 1 0 1 ST(i)

FUCOMPP = Unordered Compare Real and Pop Twice

escape 0 1 0	1 1 1 0 1 0 0 1

FTST = Test ST(0)

escape 0 0 1	1 1 1 0 0 1 0 0

FXAM = Examine ST(0)

escape 0 0 1	1 1 1 0 0 1 0 1

Constants

FLDZ = Load +0.0 into ST(0)

escape 0 0 1	1 1 1 0 1 1 1 0

FLD1 = Load +1.0 into ST(0)

escape 0 0 1	1 1 1 0 1 0 0 0

FLDPI = Load π into ST(0)

escape 0 0 1	1 1 1 0 1 0 1 1

FLDL2T = Load $\log_2 10$ into ST(0)

escape 0 0 1	1 1 1 0 1 0 0 1

FLDL2E = Load $\log_2 e$ into ST(0)

escape 0 0 1	1 1 1 0 1 0 1 0

FLDLG2 = Load $\log_{10} 2$ into ST(0)

escape 0 0 1	1 1 1 0 1 1 0 0

FLDLN2 = Load $\log_e 2$ into ST(0)

escape 0 0 1	1 1 1 0 1 1 0 1

Arithmetic

FADD = Addition

Integer/Real Memory with ST(0)

escape MF 0	mod 0 0 0 r/m

ST(i) and ST(0)

escape d P 0	1 1 0 0 0 ST(i)

FSUB = Subtraction

Integer/Real Memory with ST(0)

escape MF 0	mod 1 0 R r/m

ST(i) and ST(0)

escape d P 0	1 1 1 0 R r/m

FMUL = Multiplication

Integer/Real Memory with ST(0)

escape MF 0	mod 0 0 1 r/m

ST(i) and ST(0)

escape d P 0	1 1 0 0 1 r/m

FDIV = Division

Integer/Real Memory with ST(0)

escape MF 0	mod 1 1 R r/m

ST(i) and ST(0)

escape d P 0	1 1 1 1 R r/m

FSQRT = Square Root of ST(0)

escape 0 0 1	1 1 1 1 1 0 1 0

FSCALE = Scale ST(0) by ST(1)

escape 0 0 1	1 1 1 1 1 1 0 1

FPREM = Partial Remainder of ST(0) ÷ ST(1)

escape 0 0 1	1 1 1 1 1 0 0 0

FPREM1 = IEEE Partial Remainder

escape 0 0 1	1 1 1 1 0 1 0 1

FRNDINT = Round ST(0) to Integer

escape 0 0 1	1 1 1 1 1 1 0 0

FXTRACT = Extract Components of ST(0)

escape 0 0 1	1 1 1 1 0 1 0 0

FABS = Absolute Value of ST(0)

escape 0 0 1	1 1 1 0 0 0 0 1

FCHS = Change Sign of ST(0)

escape 0 0 1	1 1 1 0 0 0 0 0

Transcendental

FPTAN = Partial Tangent of ST(0)

escape 0 0 1	1 1 1 1 0 0 1 0

FPATAN = Partial Arctangent of ST(1) ÷ ST(0)

escape 0 0 1	1 1 1 1 0 0 1 1

FSIN = Sine

escape 0 0 1	1 1 1 1 1 1 1 0

FCOS = Cosine

escape 0 0 1	1 1 1 1 1 1 1 1

FSINCOS = Sine and Cosine

escape 0 0 1	1 1 1 1 1 0 1 1

F2XM1 = $2^{ST(0)}$ -1

escape 0 0 1	1 1 1 1 0 0 0 0

FYL2X = ST(1) x Log_2 [ST(0)]

escape 0 0 1	1 1 1 1 0 0 0 1

FYL2XP1 = ST(1) x Log_2 [ST(0) + 1]

escape 0 0 1	1 1 1 1 1 0 0 1

Processor Control

FINIT = Initialize NPX

escape 0 1 1	1 1 1 0 0 0 1 1

FSTSW AX = Store Control Word

escape 1 1 1	1 1 1 0 0 0 0 0

FLDCW = Load Control Word

escape 0 0 1	mod 1 0 1 r/m

FSTCW = Store Control Word

escape 0 0 1	mod 1 1 1 r/m

FSTSW = Store Status Word

escape 1 0 1	mod 1 1 1 r/m

FCLEX = Clear Exceptions

escape 0 1 1	1 1 1 0 0 0 1 0

FSTENV = Store Environment

escape 0 0 1	mod 1 1 0 r/m

FLDENV = Load Environment

escape 0 0 1	mod 1 0 0 r/m

FSAVE = Save State

escape 1 0 1	mod 1 1 0 r/m

FRSTOR = Restore State

escape 1 0 1	mod 1 0 0 r/m

FINCSTP = Increment Stack Pointer

escape 0 0 1	1 1 1 1 0 1 1 1

FDECSTP = Decrement Stack Pointer

escape 0 0 1	1 1 1 1 0 1 1 0

FFREE = Free ST(i)

escape 1 0 1	1 1 0 0 0 ST(i)

FNOP = No Operation

escape 0 0 1	1 1 0 1 0 0 0 0

80486 Microprocessor Instruction Set

The 80486 microprocessor uses the same instruction set that the 80386 microprocessor and the 80387 Math Coprocessor. In addition, the 80486 has six unique instructions that control cache operation:

- Byte Swap (BSWAP)
- Compare and Exchange (CMPXCHG)
- Exchange-and-Add (XADD)
- Invalidate Data Cache (INVD)
- Invalidate TLBN Entry (INVLPG)
- Write-Back and Invalidate Data Cache (WBINVD).

BSWAP = Byte Swap

0 0 0 0 1 1 1 1	1 1 0 0 1 reg

CMPXCHG = Compare and Exchange

Register 1, Register 2

0 0 0 0 1 1 1 1	1 0 1 1 0 0 0 w	1 1 reg2 reg1

Memory, Register 2

0 0 0 0 1 1 1 1	1 0 1 1 0 0 0 w	mod reg2 mem

XADD = Exchange and Add

Register 1, Register 2

00001111	1100000w	11 reg2 reg1

Memory, Register 2

00001111	1100000w	mod reg2 mem

INVD = Invalidate Data Cache

00001111	00001000

WBINVD = Write-Back and Invalidate Data Cache

00001111	00001001

INVLPG = Invalidate TLB Entry

00001111	00000001	mod 11 mem

Notes:

Direct Memory Access Controller (Type 1)

Figures

Description

The Direct Memory Access (DMA) controller allows I/O devices to transfer data directly to and from memory. This frees the system microprocessor of I/O tasks, resulting in a higher throughput.

The DMA controller is software programmable. The system microprocessor can address the DMA controller and read or modify the internal registers to define the various DMA modes, transfer addresses, transfer counts, channel masks, and page registers.

The functions of the DMA controller can be grouped into two categories: program mode and DMA transfer mode.

In the program mode, the system microprocessor accesses the DMA controller within the specific address range. These addresses are identified in Figure 1 on page 3. In this mode, the DMA registers can be read from or written to.

In the DMA mode, the DMA controller performs the data transfer. The transfer is initiated when a DMA slave has won the arbitration bus and the DMA controller has been programmed to service the winning request in process. Data transfers can be a single-byte transfer, or multiple-byte transfers (burst).

Deactivation of CD CHRDY by a device can extend accesses for slower I/O or memory devices.

The DMA controller supports the following:

- Register and program compatibility with the IBM Personal Computer AT® DMA channels (8237 compatible mode)
- 16MB (MB equals 1,048,576 bytes) 24-bit address capability for memory and 64KB (KB equals 1024 bytes) 16-bit address capability for I/O
- Eight independent DMA channels capable of transferring data between memory and I/O devices
- DMA operation with a separate read and write cycle for each transfer operation
- Channel programmable for byte or word transfer
- Extended operations:

Personal Computer AT is a registered trademark of the International Business Machines Corporation.

  - Extended program control
  - Extended Mode register
- 8- and 16-bit DMA slaves only
- Programmable arbitration levels for two channels.

DMA Controller Operations

The DMA controller does two types of operations:

- Data transfers between memory and I/O devices
- Read verifications.

Data Transfers between Memory and I/O Devices

The DMA controller performs serial transfers for all read and write operations. These transfers can be between memory and I/O on any channel. Data is read from a device and latched in the DMA controller before it is written back to a second device. The memory address needs to be specified only for a DMA data transfer. A programmable 16-bit I/O address can be provided during the I/O portion of the transfer as a programmable option. If the programmable 16-bit I/O address is not selected, the I/O address is forced to hex 0000 during the I/O transfer.

Read Verifications

The DMA controller can do a memory-read operation without a transfer. The address and the count are updated, and the terminal count is provided.

DMA I/O Address Map

Address (Hex)	Description	Bit Description	Byte Pointer
0000	Channel 0, Memory Address Register	00-15	Used
0001	Channel 0, Transfer Count Register	00-15	Used
0002	Channel 1, Memory Address Register	00-15	Used
0003	Channel 1, Transfer Count Register	00-15	Used
0004	Channel 2, Memory Address Register	00-15	Used
0005	Channel 2, Transfer Count Register	00-15	Used
0006	Channel 3, Memory Address Register	00-15	Used
0007	Channel 3, Transfer Count Register	00-15	Used
0008	Channel 0-3, Status Register	00-07	
000A	Channel 0-3, Mask Register (Set/Reset)	00-02	
000B	Channel 0-3, Mode Register (Write)	00-07	
000C	Clear Byte Pointer (Write)	N/A	
000D	DMA Controller Reset (Write)	N/A	
000E	Channel 0-3, Clear Mask Register (Write)	N/A	
000F	Channel 0-3, Write Mask Register	00-03	
0018	Extended Function Register (Write)	00-07	
001A	Extended Function Execute	00-07	Used *
0081	Channel 2, Page Table Address Register **	00-07	
0082	Channel 3, Page Table Address Register **	00-07	
0083	Channel 1, Page Table Address Register **	00-07	
0087	Channel 0, Page Table Address Register **	00-07	
0089	Channel 6, Page Table Address Register **	00-07	
008A	Channel 7, Page Table Address Register **	00-07	
008B	Channel 5, Page Table Address Register **	00-07	
008F	Channel 4, Page Table Address Register **	00-07	
00C0	Channel 4, Memory Address Register	00-15	Used
00C2	Channel 4, Transfer Count Register	00-15	Used
00C4	Channel 5, Memory Address Register	00-15	Used
00C6	Channel 5, Transfer Count Register	00-15	Used
00C8	Channel 6, Memory Address Register	00-15	Used
00CA	Channel 6, Transfer Count Register	00-15	Used
00CC	Channel 7, Memory Address Register	00-15	Used
00CE	Channel 7, Transfer Count Register	00-15	Used
00D0	Channel 4-7, Status Register	00-07	
00D4	Channel 4-7, Mask Register (Set/Reset)	00-02	
00D6	Channel 4-7, Mode Register (Write)	00-07	
00D8	Clear Byte Pointer (Write)	N/A	
00DA	DMA Controller Reset (Write)	N/A	
00DC	Channel 4-7, Clear Mask Register (Write)	N/A	
00DE	Channel 4-7, Write Mask Register	00-03	

Note: * Used only during extended functions, see "Extended Commands" on page 10. ** Upper byte of memory address register.

Figure 1. DMA I/O Address Map

Byte Pointer

A byte pointer gives 8-bit ports access to consecutive bytes of registers greater than 8 bits. For program I/O, the registers that use it are the Memory Address registers (3 bytes), the Transfer Count registers (2 bytes), and the I/O Address registers (2 bytes). Interrupts should be masked off while programming DMA controller operations.

DMA Registers

All system microprocessor access to the DMA controller must be 8-bit I/O instructions. The following figure lists the names and sizes of the DMA registers.

Register	Size (Bits)	Quantity of Registers	Allocation
Memory Address	24	8	1 per Channel
I/O Address	16	8	1 per Channel
Transfer Count	16	8	1 per Channel
Temporary Holding	16	1	All Channels
Mask	4	2	1 for Channels 7 - 4 1 for Channels 3 - 0
Arbus	4	2	1 for Channel 4 1 for Channel 0
Mode	8	8	1 per Channel
Status	8	2	1 for Channels 7 - 4 1 for Channels 3 - 0
Function	8	1	All Channels
Refresh	9	1	Independent of DMA

Figure 2. DMA Registers

Memory Address Register

Each channel has a 24-bit Memory Address register, which is loaded by the system microprocessor. The Mode register determines whether the address is incremented or decremented. The Mode register can be read by the system microprocessor in successive I/O byte operations. To read this register, the microprocessor must use the extended DMA commands.

I/O Address Register

Each channel has a 16-bit I/O Address register, which is loaded by the system microprocessor. The bits in this register do not change during DMA transfers. This register can be read by the system microprocessor in successive I/O byte operations. To read this register, the microprocessor must use the extended DMA commands.

Typically, a DMA slave is selected for DMA transfers by a decode of the arbitration level, status (-S0 exclusively ORed with -S1), and M/-IO. In this case, the respective I/O address register must have a value of 0.

A DMA slave can be selected based on a decode of the address rather than the arbitration level. In this case, the respective I/O address register must have the proper I/O address value.

Transfer Count Register

Each channel has a 16-bit Transfer Count register, which is loaded by the system microprocessor. The transfer count determines how many transfers the DMA channel will execute before reaching the terminal count. The number of transfers is always 1 more than the count specifies. If the count is 0, the DMA controller does one transfer. This register can be read by the system microprocessor in successive I/O byte operations. To read this register, the system microprocessor can use only the extended DMA commands.

Temporary Holding Register

This 16-bit register holds the intermediate value for the serial DMA transfer taking place. A DMA operation requires the data to be held in the register before it is written back. This register is not accessible by the system microprocessor.

Mask Register

Bit	Function
7 - 3	Reserved = 0
2	0 Clear Mask Bit
	1 Set Mask Bit
1, 0	00 Select Channel 0 or 4
	01 Select Channel 1 or 5
	10 Select Channel 2 or 6
	11 Select Channel 3 or 7

Figure 3. Set/Clear Single Mask Bit Using 8237 Compatible Mode

Bit	Function
7 - 4	Reserved = 0
3	0 Clear Channel 3 or 7 Mask Bit
	1 Set Channel 3 or 7 Mask Bit
2	0 Clear Channel 2 or 6 Mask Bit
	1 Set Channel 2 or 6 Mask Bit
1	0 Clear Channel 1 or 5 Mask Bit
	1 Set Channel 1 or 5 Mask Bit
0	0 Clear Channel 0 or 4 Mask Bit
	1 Set Channel 0 or 4 Mask Bit

Figure 4. DMA Mask Register Write Using 8237 Compatible Mode

Each channel has a corresponding mask bit that, when set, disables the DMA from servicing the requesting device. Each mask bit can be set to 0 or 1 by the system microprocessor. A system reset or DMA Controller Reset command sets all mask bits to 1. A Clear Mask Register command sets mask bits 0 - 3 or mask bits 4 - 7 to 0.

When a device requesting DMA cycles wins the arbitration cycle, and the mask bit is set to 1 on the corresponding channel, the DMA controller does not execute any cycles in its behalf and allows external devices to provide the transfer. If no device responds, the bus times out and causes a nonmaskable interrupt (NMI). This register can be programmed using the 8237 compatible mode commands (used by the IBM Personal Computer AT) or the extended DMA commands.

Mode Register

The Mode register for each channel identifies the type of operation that takes place when that channel transfers data.

Bit	Function
7, 6	Reserved = 0
5, 4	Reserved = 0
3, 2	00 Verify Operation 01 Write Operation 10 Read Operation 11 Reserved
1, 0	Channel Accessed 00 Select Channel 0 or 4 01 Select Channel 1 or 5 10 Select Channel 2 or 6 11 Select Channel 3 or 7

Figure 5. 8237 Compatible Mode Register

The Mode register is programmed by the system microprocessor, and its contents are reformatted and stored internally in the DMA controller. In the 8237 compatible mode, this register can only be written.

Extended Mode Register

Besides the 8237 compatible mode, all channels support an 8-bit Extended Mode register. The Extended Mode register can be programmed and read by the system microprocessor.

The DMA controller supports an Extended Mode register for each channel that can be programmed and read by the system microprocessor. This register is used whenever a DMA channel requests a DMA data transfer.

The DMA channel must be programmed to match the transfer size of the DMA slave on the channel. Bit 6 of this register is used to program the size of the DMA transfer.

Bit	Function
7	Reserved = 0
6	0 = 8-Bit Transfer 1 = 16-Bit Transfer
5	Reserved = 0
4	Reserved = 0
3	0 = Read Memory Transfer 1 = Write Memory Transfer
2	0 = Read Verifications Operation 1 = Data Transfer Operation
1	Reserved = 0
0	0 = I/O Address equals 0000H 1 = Use programmed I/O Address

Figure 6. Extended Mode Register

Status Register

The Status register, which can be read by the system microprocessor, contains information about the status of the devices. This information tells which channels have reached the terminal count and which channels have requested the bus since the last time the register was read.

Bit	Function
7	Channel 3 or 7 Request
6	Channel 2 or 6 Request
5	Channel 1 or 5 Request
4	Channel 0 or 4 Request
3	TC on Channel 3 or 7
2	TC on Channel 2 or 6
1	TC on Channel 1 or 5
0	TC on Channel 0 or 4

Figure 7. Status Register

Bits 3 through 0 in each Status register are set every time a terminal count is reached by a corresponding channel. Bits 7 through 4 are set when a corresponding arbitration level has controlled the bus. All bits are cleared by a system reset or following a system microprocessor Status Read command. This register can be read using the 8237 commands or extended DMA commands.

DMA Extended Function Register (Hex 0018)

This 8-bit register minimizes I/O address requirements and provides the extended program functions. The system microprocessor loads this register using I/O write operations. See "Extended Commands" on page 10 for more information.

Bit	Function
7 - 4	Program Command (DMA Extended Commands)
3	Reserved = 0
2 - 0	Channel Number (0 through 7)

Figure 8. DMA Extended Function Register (Hex 0018)

Arbus Register

This register is used for virtual DMA operations.

Bit	Function
7 - 4	Reserved
3 - 0	Arbitration Level

Figure 9. Arbus Register

Virtual DMA channel operation permits programming of the arbitration level assignment for channels 0 and 4 using the two 4-bit Arbus registers. These registers enable the system microprocessor to dynamically reassign the arbitration ID value by which the DMA controller responds to bus arbitration for DMA requests. This allows channels 0 and 4 to service devices at any arbitration level. The value of arbitration level hex F is reserved.

DMA Extended Operations

The function register supports an extended set of commands for the DMA channels. The extended command hex 8 programs the Arbus registers; the upper 4 bits of the Extended Function register are set to a value of 8 to select the Arbus register, and the lower 4 bits are set to the channel number (0 or 4). If channel 0 = 1 - 3 or 5 - 7 then channel 0 is active; if channel 0 = 4 then channel 0 is inactive. The system microprocessor uses the following addresses to gain control of the internal DMA registers.

I/O Address (Hex)	Command
0018	Write Extended Function Register
0019	Reserved
001A	Execute Extended Function Register
001B	Reserved

Figure 10. DMA Extended Address Decode

The system microprocessor uses the following steps to write to or read from any of the DMA internal registers:

1. Write to the Extended Function register by executing an I/O Write instruction to address hex 0018, with the proper data to indicate the function and the channel number. The internal byte pointer is always reset to 0 when an I/O write to address hex 0018 is detected.
2. Execute the Extended Function command by doing an I/O Read or I/O Write instruction to address hex 001A. The byte pointer automatically increments and points to the next byte each time port address hex 001A is used. This step is not required for Direct commands because they are executed when the Out command to address hex 0018 is detected.

Extended Commands

The following figure shows the available extended command set contained in the Extended Function register.

Registers/Bits Accessed	**Bits**	**Extended Command (Hex) (7-4 *)**	**Byte Pointer**
I/O Address Register	00-15	0	Used
Reserved		1	
Memory Address Register Write	00-23	2	Used
Memory Address Register Read	00-23	3	Used
Transfer Count Register Write	00-15	4	Used
Transfer Count Register Read	00-15	5	Used
Status Register Read	00-07	6	
Mode Register	00-07	7	
Arbus Register	00-07	8	
Mask Register Set Single Bit **		9	
Mask Register Reset Single Bit **		A	
Reserved		B	
Reserved		C	
Master Clear **		D	
Reserved		E	
Reserved		F	

Note: * Bits 7-4 of the Extended Function Register. ** Direct commands to the Extended Function register

Figure 11. DMA Extended Commands

The following is an example showing the programming of DMA channel 2 using the 8237 compatible mode and the extended mode. In this example, to perform each step, write the data indicated to the corresponding addresses.

Program Step	**8237 Compatible Mode Address/Data**	**Extended Mode Address/Data**
Set Channel Mask Bit	(000AH) x6H	(0018H) 92H
Clear Byte Pointer	(000CH) xxH	(0018H) 22H
Write Memory Address	(0004H) xxH	(001AH) xxH
Write Page Table Address	(0081H) xxH	(001AH) xxH
Clear Byte Pointer	(000CH) xxH	(0018H) 42H
Write Register Count	(0005H) xxH	(001AH) xxH
Write Register Count	(0005H) xxH	(001AH) xxH
Write Mode Register	(000BH) xxH	(0018H) 72H
		(001AH) xxH
Clear Channel 2 Mask Bit	(000AH) x2H	(0018H) A2H

Note: x's represent data.

Figure 12. DMA Channel 2 Programming Example, Extended Commands

Interrupt Controller

Figures

Description

The system provides 16 levels of hardware interrupts. Any interrupt can be masked, including the nonmaskable interrupt (NMI). The interrupt controller must be initialized to the level-sensitive mode; the edge-triggered mode is not supported. Attempts to set the controller to the edge-triggered mode will result in level-sensitive operation. For more information on nonmaskable interrupt, see the system-specific technical references.

Interrupt Assignments

The following figure shows the interrupt assignments, interrupt levels, and their functions. The interrupt levels are listed by order of priority, from highest (NMI) to lowest (IRQ 7). See system-specific technical references for masking interrupts.

Level	Master Function	Level	Slave Function
NMI	Channel Check *		
IRQ 0	Timer		
IRQ 1	Keyboard		
IRQ 2	Cascade Interrupt Control—	IRQ 8	Real Time Clock
		IRQ 9	Redirect Cascade
		IRQ 10	Reserved
		IRQ 11	Reserved
		IRQ 12	Auxiliary Device
		IRQ 13	Math Coprocessor Exception
		IRQ 14	Fixed Disk
		IRQ 15	Reserved
IRQ 3	Serial Alternate		
IRQ 4	Serial Primary		
IRQ 5	Reserved		
IRQ 6	Diskette		
IRQ 7	Parallel Port		

IRQ 8 through 15 are cascaded through IRQ 2

Note: * For channel check and other system specific functions, refer to the system-specific technical references.

Figure 1. Interrupt Level Assignments by Priority

Interrupt Sharing

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 slave interrupt controller.
3. When the interrupt occurs, the system microprocessor passes control to the IRQ9 (interrupt hex 71) interrupt handler.
4. The interrupt handler performs an end-of-interrupt (EOI) to the slave interrupt controller and passes control to the IRQ2 (interrupt hex 0A) interrupt handler.
5. The 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.

Note: Prior to the programming of the interrupt controllers, interrupts should be disabled with a CLI instruction. This includes the Mask register, EOIs, initialization command bytes, and operation command bytes.

Interrupt Controller Registers

The interrupt controller contains the following registers:

- Interrupt Request register
- In-Service register
- Interrupt Mask register
- Initialization Command registers
- Operation Command registers.

Interrupt Request Register and In-Service Register

These registers handle incoming interrupt requests. The Interrupt Request register stores all interrupt levels requesting service. The In-Service register stores all interrupt levels currently being serviced. A priority resolver prioritizes the bits in the Interrupt Request register and strobes the bit with the highest priority into the corresponding bit of the In-Service register.

Both registers can be read by issuing a Read Register command through Operation Command Byte 3, and then reading port hex 0020 or 00A0. The controller keeps track of the last register selected; therefore, subsequent reads of the same register do not require another Operation Command Byte 3 to be written. See "Operation Command Byte 3" on page 12 for more information.

Note: After initialization, the controller is set to read the Interrupt Request register.

Interrupt Mask Register

The Interrupt Mask register contains bits that mask each of the interrupt request lines of the Interrupt Request register. Lower priority levels are not affected when a higher priority level is masked.

The contents of this register are placed on the output data bus when -READ is active and port hex 0021 or 00A1 is accessed.

Initialization Command Registers and Operation Command Registers

These registers store commands from the system microprocessor that define initialization parameters and operating modes. See "Programming the Interrupt Controller" on page 8 for more information.

Modes of Operation

The interrupt controller can be programmed to operate in a variety of modes through the Initialization Command bytes and the Operation Command bytes.

Fully-Nested Mode

In the fully-nested mode, interrupts are prioritized from 0 (highest priority) to 7 (lowest priority). This mode is automatically entered after initialization unless another mode has been defined.

Note: The priorities can be changed by rotating the priorities through Operation Command Byte 2.

A typical interrupt request occurs in the following manner:

1. One or more 'interrupt request' lines are set active, causing the corresponding bits in the Interrupt Request register to be set to 1.
2. The interrupt controller evaluates the requests and sends an interrupt to the system microprocessor, if appropriate.
3. The system microprocessor responds with an 'interrupt acknowledge' pulse to the interrupt controller.
4. The controller prioritizes the unmasked bits in the Interrupt Request register and strobes the bit with the highest priority into the corresponding bit of the In-Service register. No data is sent to the system microprocessor.

   Note: If an interrupt request is not present (for example, the duration of the request was too short), the interrupt controller issues an interrupt 7.
5. The system microprocessor sends a second 'interrupt acknowledge' pulse to the interrupt controller.
6. The interrupt controller responds by releasing the interrupt vector on the data bus, where it is read by the system microprocessor.
7. The highest priority in-service bit remains set to 1 until the proper End of Interrupt command is issued by the interrupt subroutine. If the source of the interrupt request is the slave interrupt controller, the End of Interrupt command must be issued twice, once for the master and once for the slave. When the in-service bit is set to 1, all other interrupts with the same or lower priority are inhibited; interrupts with a higher priority cause an interrupt, but the interrupt is acknowledged only if the

system-microprocessor interrupt input has been re-enabled by software.

The End of Interrupt command has two forms, specific and nonspecific. The controller responds to a nonspecific End of Interrupt command by resetting the highest in-service bit of those set. In a mode that uses a fully-nested interrupt structure, the highest in-service bit set is the level that was just acknowledged and serviced. In a mode that can use other than the fully-nested interrupt structure, a specific End of Interrupt command is required to define which in-service bit to reset.

Note: An in-service bit masked by an Interrupt Mask register bit cannot be reset by a nonspecific End of Interrupt command when in the special mask mode. See "Special Mask Mode" on page 7 for more information.

Special Fully-Nested Mode

The special fully-nested mode is used when the priority in the slave interrupt controller must be preserved. This mode is similar to the normal nested mode with the following exceptions:

- When the slave's interrupt request is in service, the slave can still generate additional interrupt requests of a higher priority that are recognized by the master, and initiate interrupts to the system microprocessor.
- Upon completion of the interrupt service routine, software must send a nonspecific End of Interrupt command to the slave and read the slave's In-Service register to ensure that the interrupt just serviced was the only one generated by the slave. If the register is not empty, additional interrupts are pending, and an End of Interrupt command must not be sent to the master. If the register is empty, a nonspecific End of Interrupt command can be sent to the master.

The special fully-nested mode is selected through Initialization Command Byte 4. See "Initialization Command Byte 4" on page 10 for more information.

Automatic Rotation Mode

The automatic rotation mode accommodates multiple devices having the same interrupt priority. After a device is serviced, it is assigned the lowest priority and must wait until all other devices requesting an interrupt are serviced once before the first device is serviced again.

The following example shows the status and priorities of the In-Service register bits before and after bit 4 of the Interrupt-Request register is serviced by a Rotation on Nonspecific End of Interrupt command.

In-Service Register Bits	Status before Service	Priority before Rotate	Status after Service	Priority after Rotate
7	0	7 (Lowest)	0	2
6	1 (Pending)	6	1 (Pending)	1
5	0	5	0	0 (Highest)
4	1 (Pending)	4	0 (Serviced)	7 (Lowest)
3	0	3	0	6
2	0	2	0	5
1	0	1	0	4
0	0	0 (Highest)	0	3

Figure 2. Automatic Rotation Mode

The automatic rotation mode is selected by issuing a Rotation on Nonspecific End of Interrupt command through Operation Command Byte 2. See "Operation Command Byte 2" on page 11 for more information.

Specific Rotation Mode

The specific rotation mode allows the application programs to change the priority levels by assigning the lowest priority to a specific interrupt level. Once the lowest-level priority is selected, all other priority levels change. The following example compares the normal-nested mode to the specific rotation mode with bit 5 of the Interrupt Request register set to the lowest priority.

Interrupt Request Register Bits	Nested Mode Priority Level	Specific Rotation Mode Priority Level
7	7 (Lowest)	1
6	6	0 (Highest)
5	5	7 (Lowest)
4	4	6
3	3	5
2	2	4
1	1	3
0	0 (Highest)	2

Figure 3. Specific Rotation Mode when IRQ5 Has the Lowest Priority

The specific rotation mode is selected by issuing a Rotate on Specific End of Interrupt command or a Set Priority command through Operation Command Byte 2. See “Operation Command Byte 2” on page 11 for more information.

Special Mask Mode

The special mask mode allows application programs to selectively enable and disable any interrupt or combination of interrupts at any time during its execution. The special mask mode is selected through Operation Command Byte 3. Once the controller is in the special mask mode, setting a bit in Operation Command Byte 1 sets a corresponding bit in the Interrupt Mask register. Each bit set in the Interrupt Mask register masks the corresponding interrupt channel. Interrupt channels above and below a masked channel are not affected. See “Operation Command Byte 1” on page 11 and “Operation Command Byte 3” on page 12 for more information.

Poll Mode

Before the poll mode can be used, a CLI instruction must be issued to disable the system-microprocessor interrupt input. Devices are serviced by software issuing a Poll command through Operation Command Byte 3. The first 'read' pulse following a Poll command is interpreted by the controller as an 'interrupt acknowledge' pulse; the controller sets the appropriate in-service bit and reads the priority level. The byte placed on the data bus during a 'read' pulse is shown in the following figure.

Bit	Function
7	Interrupt Present
6-3	Undefined
2-0	Highest Priority Level

Figure 4. Poll Mode Status Byte

Bit 7 This bit is set to 1 if an interrupt is present.

Bits 6 - 3 These bits are not used and may be set to either 0 or 1.

Bits 2 - 0 These bits contain the binary code of the highest priority level requesting service.

Level-Sensitive Mode

The interrupt controller cannot be placed in the edge-triggered mode. In the level-sensitive mode, interrupt requests are recognized by a high level on the interrupt-request input. Interrupt requests must be removed before the End of Interrupt command is issued to prevent a second interrupt from occurring.

Programming the Interrupt Controller

Before the system can be used, the interrupt controller must be programmed with four sequential initialization commands. When a command is issued to a master at port hex 0020 (or a slave at port hex 00A0) with bit 4 set to 1, the command is recognized as Initialization Command Byte 1. Initialization Command Byte 1 is the first of four initialization commands required to program the interrupt controller. The following events occur during the initialization sequence:

1. The level sense circuit is set to the level-sensitive mode. (Following the initialization procedure, interrupts are generated by a high level on the interrupt-request input.)
2. The Interrupt Mask register is cleared.
3. IRQ7 is assigned priority 7.
4. The slave mode address is set to 7.
5. The special mask mode is cleared and the controller is set to read the Interrupt Request register.

Once the interrupt controller is programmed by the initialization command bytes, the controller can be programmed by the operation command bytes to operate in other modes.

Note: The master interrupt controller must be initialized before the slave interrupt controller. Failure to do so will cause unexpected results.

Initialization Command Byte 1

This is the first byte of the 4-byte initialization command sequence. This byte is issued to either the master (port hex 0020) or the slave (port hex 00A0).

Bit	Function
7 - 5	Reserved – Must be set to 0.
4	Initialization Command Byte 1 Identifier – Must be set to 1.
3	Level-Sensitive Mode – Must be set to 1.
2	Call Address Interval of 8 – Must be set to 0.
1	Cascade Mode – Must be set to 0.
0	4-byte Initialization Command Sequence – Must be set to 1.

Figure 5. Initialization Command Byte 1

Initialization Command Byte 2

This byte defines the address of the interrupt vector. Bits 7 through 3 define the five high-order bits of the interrupt vector address. Bits 2 through 0 are initialized to 0 and replaced by the hardware interrupt level when an interrupt occurs. This byte is issued to either the master (port hex 0021) or the slave (port hex 00A1).

Bit	Function
7 - 3	Bits 7 - 3 of the Interrupt Vector Address
2 - 0	Initialized to 0

Figure 6. Initialization Command Byte 2

Initialization Command Byte 3

This byte loads a value into an 8-bit slave register.

- In the master device mode, this byte has a value of hex 04 to identify interrupt 02 as a slave providing the input request.
- In the slave device mode, this byte has a value of hex 02 to tell the slave that it is using hardware interrupt 02 to communicate with the master. The slave compares this value to the cascade input; if they are equal, the slave releases the interrupt vector address on the data bus.

This byte is issued to either the master (port hex 0021) or the slave (port hex 00A1).

Bit	Master Function	Slave Function
7 - 3	0	0
2	1	0
1	0	1
0	0	0

| *Figure 7. Initialization Command Byte 3*

Initialization Command Byte 4

This byte is issued to either the master (port hex 0021) or the slave (port hex 00A1).

Bit	Function
7 - 5	Reserved – Must be set to 0.
4	Special Fully-Nested Mode
3, 2	Reserved – Must be set to 0.
1	Normal End of Interrupt – Must be set to 0.
0	80286/80386 Microprocessor Mode – Must be set to 1.

Figure 8. Initialization Command Byte 4

Operation Command Byte 1

This byte controls the individual bits in the Interrupt Mask register.

This byte is issued to either the master (port hex 0021) or the slave (port hex 00A1).

Bit	Function
7 - 0	Interrupt Mask Bits 7 - 0

Figure 9. Operation Command Byte 1

Bits 7 - 0 When set to 1, these bits inhibit their respective interrupt request input signals.

Operation Command Byte 2

This byte controls the interrupt priority and End of Interrupt command.

This byte is issued to either a master (port hex 0020) or a slave (port hex 00A0).

Bit	Function
7	Rotate Mode
6	Set Interrupt Level
5	End of Interrupt Mode
4, 3	Reserved – Must be set to 0
2 - 0	Interrupt Level (When bit 6 = 1)

Figure 10. Operation Command Byte 2

Bits 7 - 5 These bits define the rotate mode, end of interrupt mode, or a combination of the two, as shown in Figure 11 on page 12.

Bit 7	Bit 6	Bit 5	Function
0	0	0	Reserved
0	0	1	Nonspecific End of Interrupt Command
0	1	0	No Operation
0	1	1	Specific End of Interrupt Command*
1	0	0	Reserved
1	0	1	Rotate on Nonspecific End of Interrupt Command
1	1	0	Set Priority Command**
1	1	1	Rotate on Specific End of Interrupt Command**

Note: * Bits 0, 1, and 2 are the binary level of the in-service bit to be reset. ** Bits 0, 1, and 2 are the binary level of the lowest-priority device.

Figure 11. Operation Command Byte 2 (Bits 7 - 5)

Bits 4, 3 These bits are reserved and must be set to 0.

Bits 2 - 0 These bits define the hardware interrupt level to be acted upon when bit 6 is set to 1.

Bit 2	Bit 1	Bit 0	Function
0	0	0	Interrupt Level 0
0	0	1	Interrupt Level 1
0	1	0	Interrupt Level 2
0	1	1	Interrupt Level 3
1	0	0	Interrupt Level 4
1	0	1	Interrupt Level 5
1	1	0	Interrupt Level 6
1	1	1	Interrupt Level 7

Figure 12. Operation Command Byte 2 (Bits 2 - 0)

Operation Command Byte 3

This byte is issued to either a master (port hex 0020) or a slave (port hex 00A0).

Bit	Function
7	Reserved – Must be set to 0.
6, 5	Special Mask Mode Bits
4	Reserved – Must be set to 0.
3	Reserved – Must be set to 1.
2	Poll Command
1, 0	Read Register Command

Figure 13. Operation Command Byte 3

Bit 7 This bit is reserved and must be set to 0.

Bits 6, 5 These bits enable the special mask mode, as shown in the following figure.

Bit 6	Bit 5	Function
0	0	No Action
0	1	No Action
1	0	Normal Mask Mode
1	1	Special Mask Mode

Figure 14. Operation Command Byte 3 (Bits 6 and 5)

Bit 4 This bit is reserved and must be set to 0.

Bit 3 This bit is reserved and must be set to 1.

Bit 2 When set to 1, this bit sets the Poll command.

Bits 1, 0 These bits determine the register to be read on the next 'read' pulse, as shown in the following figure.

Bit 1	Bit 0	Function
0	0	No Action
0	1	No Action
1	0	Read Interrupt Request Register
1	1	Read In-Service Register

Figure 15. Operation Command Byte 3 (Bits 1 and 0)

Notes:

System Timers (Type 1)

Figures

Description

The system has three programmable timers/counters: Channel 0 is the System Timer, Channel 2 is the Tone Generator for the speaker, and Channel 3 is the Watchdog Timer. Channel 0 and Channel 2 are similar to Channel 0 and Channel 2 of the IBM Personal Computer, IBM Personal Computer XT™, and the IBM Personal Computer AT®. Channel 3 does not have a counterpart in earlier IBM personal computer systems. The following is a block diagram of the counters.

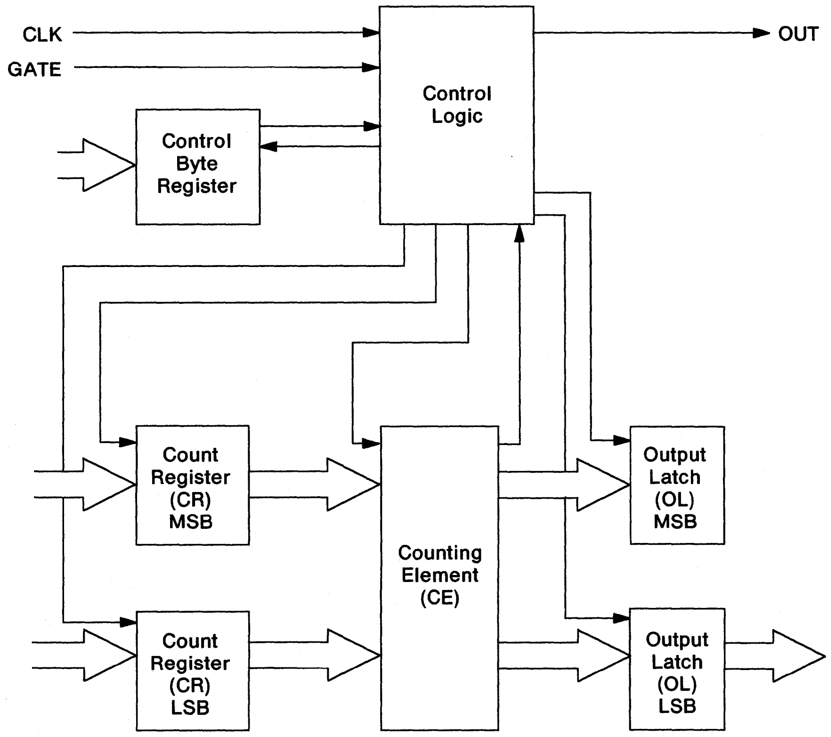


Figure 1. Counters

Personal Computer XT is a trademark of the International Business Machines Corporation.

Personal Computer AT is a registered trademark of the International Business Machines Corporation.

Channel 0 - System Timer

- GATE 0 is always enabled.
- CLK IN 0 is driven by a 1.193 MHz signal.
- CLK OUT 0 indirectly drives the ‘interrupt request 0’ signal (IRQ 0).

  IRQ 0 is driven by a latch that is set by the rising edge of the ‘clock out 0’ signal (CLK OUT 0). The latch can be cleared by a system reset, an interrupt acknowledgment cycle with a vector of hex 08, or an I/O write to System Control Port B (hex 0061) setting bit 7 to 1.

  Signals derived from CLK OUT 0 are used to gate and clock Channel 3.

Channel 2 - Tone Generation for Speaker

- GATE 2 is controlled by bit 0 of port hex 0061.
- CLK IN 2 is driven by a 1.193 MHz signal.
- CLK OUT 2 has two connections. One is to input port hex 0061, bit 5. CLK OUT 2 is also logically ANDed with port hex 0061, bit 1 (speaker data enable). The output of the AND gate drives the ‘audio sum node’ signal.

The audio subsystem is a speaker driven by a linear amplifier. The linear amplifier input node can be driven from the following sources:

- System-timer Channel 2 when enabled using bit 1 of I/O port hex 0061 set to 1. (For information about system timer Channel 2 see "Description" on page 1.)
- The system channel using the 'audio sum node' signal.

The following block diagram shows the audio subsystem.

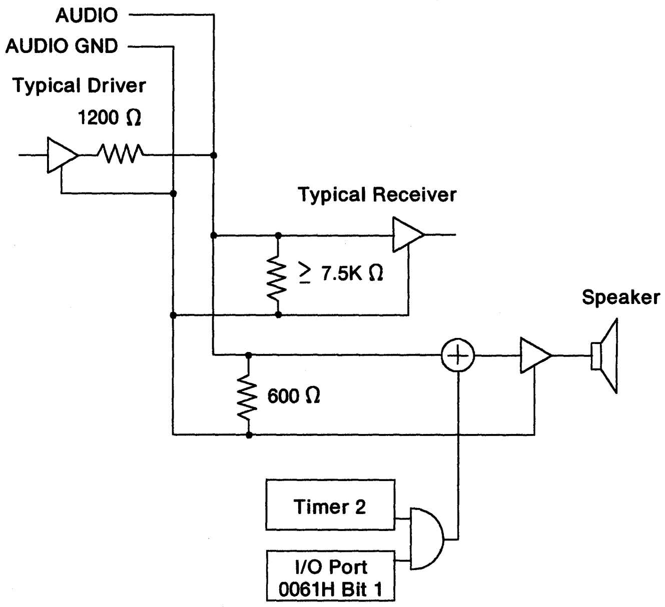


Figure 2. Audio Subsystem Block Diagram

Each audio driver must have a 1200 ohm source impedance, and a 7.5 kilohm or greater impedance is required for each audio receiver. Volume control is provided by the driver. Output level is a function of the number of drivers and receivers that share the AUDIO line.

The logic ground is connected to AUDIO GND at the amplifier.

Channel 3 - Watchdog Timer

This channel operates only in Mode 0 and counts in 8-bit binary.

- GATE 3 is tied to IRQ 0.
- CLK IN 3 is tied to CLK OUT 0 inverted.
- CLK OUT 3, when high, drives the NMI active.

The Watchdog Timer detects when IRQ 0 is active for more than one period of CLK OUT 0. If IRQ 0 is active when a rising edge of CLK OUT 0 occurs, the count is decremented. When the count is decremented to 0, an NMI is generated. Thus, the Watchdog Timer can be used to detect when IRQ 0 is not being serviced. This is useful for detecting error conditions.

BIOS interfaces are provided to enable and disable the Watchdog Timer. When the Watchdog Timer times out, it causes an NMI and sets System Control Port A (hex 0092), bit 4 to 1. This bit may be set to 0 by using the BIOS interface to disable the Watchdog Timer.

Note: The NMI stops all arbitration on the bus until bit 6 of the Arbitration register (I/O address hex 0090) is set to 0. This can result in lost data or an overrun error on some I/O devices.

If the Watchdog Timer is used to detect "tight looping" software tasks that inhibit interrupts, some I/O devices may be overrun (not serviced in time). The operating system may be required to restart these devices.

When the Watchdog Timer is enabled, the 'inhibit' signal (INHIBIT) is active only when IRQ 0 is pending for longer than one period of CLK OUT 0. When INHIBIT is active, any data written to Channel 0 or Channel 3 is ignored. INHIBIT is never active if the Watchdog Timer is disabled.

The Watchdog Timer operation is defined only when Channel 0 is programmed in Mode 2 or Mode 3. The operation of the Watchdog Timer is undefined when Channel 0 is programmed in any other mode.

Counters 0, 2, and 3

Each counter is independent. Counters 0 and 2 are 16-bit down counters that can be preset. They can count in binary or binary coded decimal (BCD). Counter 3 is an 8-bit down counter that can be preset. It counts in binary only.

Programming the System Timers

The system treats the programmable interval timer as an arrangement of five external I/O ports. Three ports are treated as count registers and two are control registers for mode programming. Counters are programmed by writing a control word and then an initial count. All control words are written into the Control Word registers, which are located at address hex 0043 for counters 0 and 2, and address hex 0047 for counter 3. Initial counts are written into the Count registers, not the Control Byte registers. The format of the initial count is determined by the control word used.

After the initial count is written to the Count register, it is transferred to the counting element, according to the mode definition. When the count is read, the data is presented by the output latch.

Counter Write Operations

The control word must be written before the initial count, and the count must follow the count format specified in the control word.

A new initial count may be written to the counters at any time without affecting the counter′s programmed mode. Counting is affected as described in the mode definitions. The new count must follow the programmed count format.

Counter Read Operations

The counters can be read using the Counter Latch command (see “Counter Latch Command” on page 10).

If the counter is programmed for two-byte counts, two bytes must be read. The two bytes need not be read consecutively; read, write, or programming operations of other counters can be inserted between them.

Note: If the counters are programmed to read or write two-byte counts, the program must not transfer control between writing the first and second byte to another routine that also reads or writes into the same counter. This will cause an incorrect count.

Registers

I/O Address (Hex)	Register
0040	Count Register - Channel 0 (Read/Write)
0042	Count Register - Channel 2 (Read/Write)
0043	Control Byte Register - Channel 0 or 3 (Write)
0044	Count Register - Channel 3 (Read/Write)
0047	Control Byte Register - Channel 3 (Write)

Figure 3. System Timer/Counter Registers

Count Register - Channel 0 (Hex 0040)

The control byte is written to port hex 0043, to indicate the format of the count (least-significant byte only, most-significant byte only, or least-significant byte followed by most-significant byte). This must be done before writing the count to port hex 0040.

Count Register - Channel 2 (Hex 0042)

The control byte is written to port hex 0043, to indicate the format of the count (least-significant byte only, most-significant byte only, or least-significant byte followed by most-significant byte). This must be done before writing the count to port hex 0042.

Control Byte Register - Channel 0 or 2 (Hex 0043)

This is a write-only register. The following gives the format for the control byte (port hex 0043) for counters 0 and 2.

Bits 7, 6 These bits select counter 0 or 2.

Bits 7 6	Function
0 0	Select Counter 0
0 1	Reserved
1 0	Select Counter 2
1 1	Reserved

Figure 4. Select Counter Bits, Port Hex 0043

Bits 5, 4 These bits distinguish a counter latch command from a control byte. If a control byte is selected, these bits also determine the method in which each byte is read or written.

Bits 5 4	Function
0 0	Counter Latch Command
0 1	Read/Write Counter bits 0 - 7 only
1 0	Read/Write Counter bits 8 - 15 only
1 1	Read/Write Counter bits 0 - 7 first, then bits 8 - 15

Figure 5. Read/Write Counter Bits, Port Hex 0043

Bits 3 - 1 These bits select the mode.

Bits 3 2 1	Function
0 0 0	Mode 0 - Interrupt on Terminal Count
0 0 1	Mode 1 - Hardware Retriggerable One Shot
X 1 0	Mode 2 - Rate Generator
X 1 1	Mode 3 - Square Wave
1 0 0	Mode 4 - Software Retriggerable Strobe
1 0 1	Mode 5 - Hardware Retriggerable Strobe

Note: Don't care bits (X) should be set to 0.

Figure 6. Counter Mode Bits, Port Hex 0043

Bit 0 When set to 1, this bit selects the binary coded decimal method of counting. When set to 0, it selects the 16-bit binary method.

Count Register - Channel 3 (Hex 0044)

The control byte is written to port hex 0047, to indicate the format of the count (least-significant byte only). This must be done before writing the count to port hex 0044.

Control Byte Register - Channel 3 (Hex 0047)

This is a write-only register. The following gives the format for the control byte (port hex 0047) for counter 3.

Bits 7, 6 These bits select counter 3.

Bits 7 6	Function
0 0	Select Counter 3
0 1	Reserved
1 0	Reserved
1 1	Reserved

Figure 7. Select Counter, Port Hex 0047

Bits 5, 4 These bits distinguish a counter latch command from a control byte.

Bits 5 4	Function
0 0	Counter Latch Command Select Counter 0
0 1	R/W Counter Bits 0 - 7 Only
1 0	Reserved
1 1	Reserved

Figure 8. Read/Write Counter, Port Hex 0047

Bits 3 - 0 These bits are reserved and must be written as 0.

Counter Latch Command

The Counter Latch command is written to the Control Byte register. Bits 7 and 6 select the counter, and bits 5 and 4 distinguish this command from a control byte. The following figure shows the format of the Counter Latch command.

Bit	Function
7, 6	Specifies the counter to be latched
5, 4	00 Specifies the Counter Latch command
3 - 0	Reserved = 0

Figure 9. Counter Latch Command

The count is latched into the selected counter's output latch when the Counter Latch command is received. This count is held in the latch until it is read by the system microprocessor (or until the counter is reprogrammed). After the count is read by the system microprocessor, it is automatically unlatched, and the output latch returns to following the counting element. Counter Latch commands do not affect the programmed mode of the counter in any way. All subsequent latch commands issued to a given counter before the count is read, are ignored. A read cycle to the counter latch returns the value latched by the first Counter Latch command.

System Timer Modes

The following definitions are used when describing the timer modes.

CLK pulse	A rising edge, then a falling edge on the counter CLK input.
Trigger	A rising edge on a counter's input GATE.
Counter Load	The transfer of a count from the Counter register to the counting element.

Mode 0 - Interrupt on Terminal Count

Event counting can be done using Mode 0. Counting is enabled when GATE is equal to 1, and disabled when GATE is equal to 0. If GATE is equal to 1 when the control byte and initial count are written to the counter, the sequence is as follows:

1. The control byte is written to the counter, and OUT goes low.
2. The initial count is written.
3. The initial count is loaded on the next CLK pulse. The count is not decremented for this CLK pulse.

   The count is decremented until the counter reaches 0. For an initial count of N, the counter reaches 0 after N+1 CLK pulses.
4. OUT goes high.

OUT remains high until a new count or new Mode 0 control byte is written into the counter.

If GATE equals 0 when an initial count is written to the counter, it is loaded on the next CLK pulse even though counting is not enabled. After GATE enables counting, OUT goes high N CLK pulses later.

If a new count is written to a counter while counting, it is loaded on the next CLK pulse. Counting then continues from the new count. If a 2-byte count is written to the counter, the following occurs:

1. The first byte written to the counter disables the counting. OUT goes low immediately, and there is no delay for the CLK pulse.
2. When the second byte is written to the counter, the new count is loaded on the next CLK pulse. OUT goes high when the counter reaches 0.

Mode 1 - Hardware Retriggerable One-Shot

The sequence for Mode 1 is as follows:

1. OUT is high.
2. On the CLK pulse following a trigger, OUT goes low and begins the one-shot pulse.
3. When the counter reaches 0, OUT goes high.

OUT remains high until the CLK pulse following the next trigger.

The counter is armed by writing the control byte and initial count to the counter. When a trigger occurs, the counter is loaded. OUT goes low on the next CLK pulse, starting the one-shot pulse. For an initial count of N, a one-shot pulse is N CLK pulses long. The one-shot pulse repeats the count of N for the next triggers. OUT remains low for N CLK pulses following any trigger. GATE does not affect OUT. The current one-shot pulse is not affected by a new count written to the counter, unless the counter is retriggered. If the counter is retriggered, the new count is loaded and the one-shot pulse continues.

Note: Mode 1 is valid only for Counter 2.

Mode 2 - Rate Generator

This mode causes the counter to perform a divide-by-N function. Counting is enabled when GATE equals 1, and disabled when GATE equals 0.

The sequence for Mode 2 is as follows:

1. OUT is high.
2. The initial count decrements to 1.
3. OUT goes low for one CLK pulse.
4. OUT goes high.
5. The counter reloads the initial count.
6. The process is repeated.

If GATE goes low during the OUT pulse, OUT goes high. On the next CLK pulse a trigger reloads the counter with the initial count. OUT goes low N CLK pulses after the trigger. This allows the GATE input to be used to synchronize the counter.

The counter is loaded on the CLK pulse after a control byte and initial count are written to the counter. OUT goes low N CLK pulses after the initial count is written. This allows software to synchronize the counter.

The current counting sequence is not affected by a new count being written to the counter. If the counter receives a trigger after a new count is written and before the end of the current count, the new count is loaded on the next CLK pulse, and counting continues from the new count. If the trigger is not received by the counter, the new count is loaded following the current counting cycle.

Mode 3 - Square Wave

Mode 3 is similar to Mode 2 except for the duty cycle of OUT. Counting is enabled when GATE is equal to 1, and disabled when GATE is equal to 0. An initial count of N results in a square wave on OUT. The period of the square wave is N CLK pulses. If OUT is low and GATE goes low, OUT goes high. On the next CLK pulse, a trigger reloads the counter with the initial count.

The counter is loaded on the CLK pulse following the writing of a control byte and the initial count.

The current counting sequence is not affected by a new count being written to the counter. If the counter receives a trigger after a new count is written, and before the end of the current count's half-cycle of the square wave, the new count is loaded on the next CLK pulse, and counting continues from the new count. If the trigger is not received by the counter, the new count is loaded following the current half-cycle.

The way Mode 3 is implemented depends on whether the count written is odd or even. If the count is even, OUT begins high and the following applies:

1. The initial count is loaded on the first CLK pulse.
2. The count is decremented by 2 on succeeding CLK pulses.
3. The count decrements to 0.
4. OUT changes state.
5. The counter is reloaded with the initial count.
6. The process repeats indefinitely.

If the count is odd, the following applies:

1. OUT is high.
2. The initial count minus 1 is loaded on the first CLK pulse.
3. The count is decremented by 2 on succeeding CLK pulses.
4. The count decrements to 0.
5. One CLK pulse after the count reaches 0, OUT goes low.
6. The counter is reloaded with the initial count minus 1.
7. Succeeding CLK pulses decrement the count by 2.
8. The count decrements to 0.
9. OUT goes high.
10. The counter is reloaded with the initial count minus 1.
11. The process repeats indefinitely.

Mode 3, using an odd count, causes OUT to go high for a count of (N+1)/2 and low for a count of (N-1)/2.

Mode 3 can operate such that OUT is initially set low when the control byte is written. For this condition, Mode 3 operates as follows:

1. OUT is low.
2. The count decrements to half of the initial count.
3. OUT goes high.
4. The count decrements to 0.
5. OUT goes low.
6. The process repeats indefinitely.

This process results in a square wave with a period of N CLK pulses.

Note: If OUT needs to be high after the control byte is written, the control byte must be written twice. This applies only to Mode 3.

Mode 4 - Software Retriggerable Strobe

Counting is enabled when GATE equals 1, and disabled when GATE equals 0. Counting begins when an initial count is written.

The sequence for Mode 4 is as follows:

1. OUT is high.
2. The control byte and initial count are written to the counter.
3. The initial count is loaded on the next CLK pulse. The count is not decremented for this clock pulse.
4. The count is decremented to 0. For an initial count of N, the counter reaches 0 after N+1 CLK pulses.
5. OUT goes low for one CLK pulse.
6. OUT goes high.

GATE should not go low one-half CLK pulse before or after OUT goes low. If this occurs, OUT remains low until GATE goes high.

If a new count is written to a counter while counting, it is loaded on the next CLK pulse. Counting then continues from the new count. If a 2-byte count is written, the following occurs:

1. Writing the first byte does not affect counting.
2. The new count is loaded on the CLK pulse following the writing of the second byte.

The Mode 4 sequence can be retriggered by software. The period from when the new count of N is written to when OUT strobes low is (N+1) pulses.

Mode 5 - Hardware Retriggerable Strobe

The sequence for Mode 5 is as follows:

1. OUT is high.
2. The control byte and initial count are written to the counter.
3. Counting is triggered by a rising edge of GATE.
4. The counter is loaded on the CLK pulse following the trigger. This CLK pulse does not decrement the count.
5. The count decrements to 0.
6. OUT goes low for one CLK pulse. This occurs (N+1) CLK pulses after the trigger.
7. OUT goes high.

The counting sequence can be retriggered. OUT strobes low (N+1) pulses after the trigger. GATE does not affect OUT.

The current counting sequence is not affected by a new count being written to the counter. If the counter receives a trigger after a new count is written and before the end of the current count, the new count is loaded on the next CLK pulse, and counting continues from the new count.

Note: Mode 5 is valid only on counter 2.

Operations Common to All Modes

Control bytes written to a counter cause all control logic to reset. OUT goes to a known state. This does not take a CLK pulse.

The falling edge of the CLK pulse occurs when new counts are loaded and counters are decremented.

Counters do not stop when they reach 0. In Modes 0, 1, 4, and 5, the counter wraps to the highest count, and continues counting. Modes 2 and 3 are periodic; the counter reloads itself with the initial count and continues from there.

The GATE is sampled on the rising edge of the CLK pulse.

The following shows the minimum and maximum initial counts for the counters.

Mode	Minimum Count	Maximum Count
0	1	0 = 2^{16} (Binary Counting) or 10^4 (BCD Counting)
1	1	0 = 2^{16} (Binary Counting) or 10^4 (BCD Counting)
2	2	0 = 2^{16} (Binary Counting) or 10^4 (BCD Counting)
3	2	0 = 2^{16} (Binary Counting) or 10^4 (BCD Counting)
4	1	0 = 2^{16} (Binary Counting) or 10^4 (BCD Counting)
5	1	0 = 2^{16} (Binary Counting) or 10^4 (BCD Counting)

Figure 10. Minimum and Maximum Initial Counts, Counters 0, 2

Counter 3 can use only Mode 0, Interrupt on Terminal Count. The minimum initial count is 1 and the maximum is hex FF.

Diskette Drive Controller

Figures

Description

The diskette drive controller and interface connector reside on the system board. The controller can be a Type 1 or a Type 2. The controller type is dependent on the system model and can be determined by issuing the Unlock command. The Type 1 returns a hex 80 in the Result byte; the Type 2 returns a hex 00.

The Type 1 controller supports:

- Two data transfer rates:
  - 250,000 bits per second (bps)
  - 500,000 bits per second.
- 125 nanoseconds of precompensation on all tracks
- The following IBM diskette drives:
  - 3.5-inch 1.44MB
  - 5.25-inch 360KB.

The Type 2 controller supports:

- Four data transfer rates:
  - 250,000 bits per second
  - 300,000 bits per second
  - 500,000 bits per second
  - 1,000,000 bits per second.
- Programmable precompensation
- 16 bytes of data buffering
- The following IBM diskette drives:
  - 3.5-inch 1.44MB
  - 3.5-inch 2.88MB
  - 5.25-inch 360KB
  - 5.25-inch 1.2MB.

The media formats supported by IBM drives are shown below.

Media Size	Capacity		Sectors/ Track	No. of Tracks	Data Rate (kbps)
	Unformatted	Formatted			
3.5 in.	1.0MB	720KB	9	80	250
3.5 in.	2.0MB	1.44MB	18	80	500
3.5 in.	4.0MB	2.88MB	36	80	1000*
5.25 in.	0.5MB	360KB	9	40	300* /250
5.25 in.	1.6MB	1.20MB*	15	80	500
* Type 2 controller only					

Figure 1. Media Format Table

Warning: The controller does not check to see that the media supports the selected capacity. Attempting to format media to an unsupported capacity may cause loss of data.

- 0.5MB (double sided, double density 5.25-inch) media can be reliably formatted only to the 360KB capacity.
- 1.0MB media can be reliably formatted only to the 720KB capacity.
- 1.6MB (high capacity 5.25-inch) media can be reliably formatted only to the 1.2MB capacity.
- 2.0MB media can be reliably formatted only to the 1.44MB capacity.
- 4.0MB media can be reliably formatted only to the 2.88MB capacity.

When the Type 1 diskette drive controller is switched from one data rate to another, the controller clock rate automatically changes to:

- 8 MHz for 2.0MB mode
- 4 MHz for 1.0MB mode.

The step rate time (SRT), head load time (HLT), and the head unload time (HUT) parameters must then be changed to maintain the desired timings at the diskette interface.

For Type 2 controllers, the controller clock rate remains at 24 MHz, but the SRT, HLT, and HUT parameters must still be updated using the same values as the Type 1 controller.

Note: With Type 1 controllers, 32-bit operations to the video subsystem can cause a Direct Memory Access (DMA) overrun. If the BIOS returns an error code indicating that an overrun has occurred, the operation should be repeated.

FIFO Mode

A 16-byte FIFO is provided on the Type 2 controller, which allows direct-memory-access (DMA) data transfers to be delayed for longer periods of time without causing DMA overrun errors. To maintain compatibility, the FIFO defaults to a Type 1 compatibility mode after system reset. All functions are then similar to the Type 1 controller.

The Configure command is used to enable FIFO operations. After the FIFO operations are enabled, the controller temporarily enters a byte mode during the command and result phases of the diskette controller operation. While in this mode, operations to, or from, the disk controller are FIFO compatible.

The FIFO is enabled only during the data transfer phase of operation. All command and status information is transferred in Type 1 compatibility mode. When the Type 2 controller first enters the data transfer phase, the FIFO is cleared of any residual data from previous operations.

Compatibility problems may occur when the FIFO mode is used with software that monitors the progress of a data transfer during the execution phase. It is recommended that the FIFO mode be disabled when software of this type is used.

Diskette Drive Controller Registers

The diskette drive controller can be a Type 1 and a Type 2 controller. Certain functions are provided by the Type 2 controller only.

Note: Some registers contain bits that are labeled as reserved. These bits must be set to the value specified when writing to these registers.

Status Register A (Hex 03F0)

This read-only register shows the status of signals on the diskette drive interface.

Bit	Function
7	Interrupt Pending
6	-2nd Drive Installed
5	Step
4	-Track 0
3	Head 1 Select
2	-Index
1	-Write Protect
0	Direction In

Figure 2. Status Register A (Hex 03F0)

Status Register B (Hex 03F1)

This read-only register shows the status of signals on the diskette drive interface.

The write-data and read-data bits change state for each positive transition of the '-write data' or '-read data' signals.

Bit	Function
7, 6	Reserved
5	Drive Select 0
4	Write Data
3	Read Data
2	Write Enable
1	Motor Enable 1
0	Motor Enable 0

Figure 3. Status Register B (Hex 03F1)

Drive Control Register (Hex 03F2)

This read and write register controls drive motors, drive selection, and feature enable. All bits are set to 0 by a reset.

Bit	Function
7	Motor Enable 3*
6	Motor Enable 2*
5	Motor Enable 1
4	Motor Enable 0
3	Reserved = 1
2	-Controller Reset
1	Drive Select 1*
0	Drive Select 0

* Type 2 Controller Only.

Figure 4. Drive Control Register (Hex 03F2)

Note: The controller reset bit must be set to 0 for a minimum of 3.5 microseconds to ensure the controller is properly reset.

Drive Status Register (Hex 03F3 Read)

This register contains information about the diskette drive type, and start-up drive location. This register is for the Type 2 controller only.

Bit	Function
7,6	Media Type 1,0
5,4	Drive Type 1,0
3,2	Start-Up Drive 1,0
1,0	Reserved

Figure 5. Drive Status Register (Hex 03F3 Read)

The following tables show the media type, drive type and start-up drive indicated.

Bits 7 6	**Media Type**
0 0	Reserved
0 1	4MB
1 0	2MB
1 1	1MB

Note: These bits indicate the state of the 'media type' signals. These signals are inputs from the drive.

Figure 6. Media Type

Bits 5 4	**Drive Type**
0 0	3.5-inch, 1.44MB
0 1	3.5-inch, 2.88MB
1 0	5.25 inch, 1.2MB
1 1	Reserved

Figure 7. Drive Type

Bits 3 2	**Start-Up Drive**
0 0	First Drive
0 1	Second Drive
1 0	Third Drive
1 1	Reserved

Figure 8. Start-Up Drive

Note: The media-type and drive-type bits are valid only when the drive is selected. The media-type bits are valid only for 3.5-inch media.

Diskette Drive Controller Status Register (Hex 03F4 Read)

This read-only register facilitates the transfer of data between the system microprocessor and the controller.

Bit	Function
7	Request for Master
6	Data Input/Output
5	Non-DMA Mode
4	Diskette Controller Busy
3	Drive 3 Busy
2	Drive 2 Busy
1	Drive 1 Busy
0	Drive 0 Busy

Figure 9. Diskette Drive Controller Status Register (Hex 03F4)

Bit 7	When this bit is 1, the Data register is ready to transfer data with the system microprocessor.
Bit 6	This bit indicates the direction of data transfer between the diskette drive controller and the system microprocessor. When this bit is 1, the transfer is to the system microprocessor; when the bit is 0, the transfer is to the controller.
Bit 5	When this bit is 1, the controller is in the non-DMA mode.
Bit 4	When this bit is 1, command is being processed.
Bit 3	When this bit is 1, diskette drive 3 is in the seek mode.
Bit 2	When this bit is 1, diskette drive 2 is in the seek mode.
Bit 1	When this bit is 1, diskette drive 1 is in the seek mode.
Bit 0	When this bit is 1, diskette drive 0 is in the seek mode.

Precompensation Select Register (Hex 03F4 Write)

This write-only register is used to select the data rate and precompensation value for each data rate. This register is supported by the Type 2 controllers only.

Bit	Function
7-5	Reserved = 0
4	Precomp 2
3	Precomp 1
2	Precomp 0
1	Data Rate Select 1
0	Data Rate Select 0

Figure 10. Precompensation Select Register (Hex 03F4 Write)

The following table shows the precompensation values selected.

Bits **4 3 2**	**Precompensation Delay**
0 0 0	Default
0 0 1	41.7 ns
0 1 0	83.3 ns
0 1 1	125 ns
1 0 0	167 ns
1 0 1	208 ns
1 1 0	250 ns
1 1 1	0.0 ns

Figure 11. Precompensation Values

| The following table shows the transfer rate selected and the resulting | default precompensation values.

Bits 1 0	Transfer Rate	Default Precomp
0 0	500 kbps	125 ns
0 1	300 kbps*	125 ns
1 0	250 kbps	125 ns
1 1	1000 kbps*	41.7 ns

* Type 2 Controller Only

Figure 12. Default Precompensation Values

Diskette Drive Controller Command/Data Register (Hex 03F5)

This read and write register passes data, commands and parameters to the diskette drive controller and provides diskette-drive status information.

Reserved Register (Hex 03F6)

This register is reserved.

Data Rate Status Register (Hex 03F7 - Read)

This read-only register identifies the data rate selected on the interface, and senses the state of the 'diskette change' and '-high density select' signals.

Bit	Function
7	Diskette Change
6-3	Reserved
2	Data Rate Select 1
1	Data Rate Select 0
0	-High Density Select

Figure 13. Data Rate Status Register (Hex 03F7 - Read)

Data Rate Control Register (Hex 03F7 - Write)

This write-only register sets the transfer rate.

Bit	Function
7-2	Reserved
1	Data Rate Select 1
0	Data Rate Select 0

Figure 14. Data Rate Control Register (Hex 03F7 - Write)

Bits 7 - 2 Reserved as 0.

Bits 1, 0 These bits select the data rate, as shown in the following figure.

Bits 1 0	Data Rate
0 0	500,000 bits per second
0 1	300,000 bits per second*
1 0	250,000 bits per second
1 1	1,000,000 bits per second*

* Type 2 Controller Only

Figure 15. Data Rate Selection

Diskette Drive Controller Programming Considerations

Each command is initiated by a multibyte transfer from the system microprocessor; the result can be a multibyte transfer back to the system microprocessor. Most transfers consist of three phases:

- *Command Phase:* The system microprocessor writes a series of command bytes to the controller directing it to perform a specific operation.
- *Execution Phase:* The controller performs the specified operation.
- *Result Phase:* After the operation is complete, status information is available to the system microprocessor through a sequence of read commands.

Note: The Seek, Relative Seek, and Recalibrate commands have no result phase. After issuing these commands, the Sense Interrupt Status command must be issued for proper termination and verification of the head position (Present Cylinder Number parameter or PCN).

Controller Commands

The following are supported commands for diskette drive controller:

- Configure#
- Dumpreg#
- Format Track
- Read Data
- Read Deleted Data
- Read ID
- Read Track
- Recalibrate
- Relative Seek#
- Scan Equal
- Scan High or Equal
- Scan Low or Equal
- Seek
- Sense Drive Status
- Sense Interrupt Status
- Specify
- Verify#
- Version#
- Write Data
- Write Deleted Data.

Notes:

1. Commands marked with a pound sign are not supported by the Type 1 controller.
2. Diskette BIOS does not support all commands listed. To ensure software compatibility across system models, the diskette hardware should be accessed only through the diskette BIOS.

Symbol	Name	Description
D0 - D3	Drive Perpendicular	These bits represent the state of drive 0 through drive 3. When the bit is 1, the associated drive is a perpendicular drive. GAP and WGATE must not be set to 1 for these bits to be valid.
DIR	Direction Control	When set to 0, this bit causes the head to move away from the spindle during an implied seek. When set to 1, the head moves toward the spindle.
EC	Enable Count	When this bit is set to 1, the data-length byte indicates the number of sectors/track.
EFF	Enable FIFO	When set to 1, this bit enables the FIFO data buffer in the Type 2 controller.
EIS	Enable Implied Seek	When set to 1, this bit causes an implied seek to be performed before executing a command that requires the cylinder parameter in the command phase in the Type 2 controller.
GAP	GAP 2 Size	This parameter is used with WGATE to set the length of the GAP 2 written during write or format operations. GAP should be set to 0 for normal operation.
GPL	GAP 3 Length	This parameter determines the length of GAP 3 (in bytes) during a Format operation and determines the VCO synchronization timing during Read operations.
HD	Head Select	This bit selects the head used. When set to 0, head 0 is selected; when set to 1, head 1 is selected.
HLT	Head Load Time	This parameter specifies the head-load time (see Figure 17 on page 14).
HUT	Head Unload Time	This parameter specifies the head-unload time (see Figure 18 on page 14).
MFM	FM or MFM	When set to 0, this bit selects FM mode; when set to 1, it selects MFM mode.
MT	Multitrack	When set to 1, this bit selects multitrack operations. (Side 0 and Side 1 are automatically read from, written to, or verified).

Symbol	Name	Description
ND	Non-DMA Mode	When set to 1, this bit causes diskette operations to be performed in the non-DMA mode.
OW	Overwrite	When set to 1, this bit allows the bits specified by D0-D3 to be overwritten. **Warning:** To avoid the loss of data, this bit should not be set to 1.
PD	Polling Disable	When set to 1, this bit disables polling; when set to 0, it enables polling.
PTN	Precomp Track Number	This parameter selects the track number where write precompensation begins. It is programmable from 0 to hex FF.
RCN	Relative Cylinder Number	This parameter specifies the relative cylinder to go to from the present cylinder, as used by the Relative Seek command.
SK	Skip Flag	When this bit is set to 1, the sectors containing a deleted-data address mark are automatically skipped during a Read Data command. For a Read Deleted Data command, only sectors with a deleted-data address mark are accessed. When this bit is set to 0, the sector is read or written the same as the Read Data command.
SRT	Step Rate Time	This parameter specifies the stepping-rate time (see Figure 19 on page 15).
THR	Threshold	This parameter is the FIFO threshold for Type 2 controllers, where 0 = 1 byte, F = 16 bytes.
US	Unit Select	This parameter indicates the drive number selected. 00 selects drive 0, 01 selects drive 1, 10 selects drive 2 and 11 selects Drive 3.
WGATE	Write Gate	This parameter alters the timing of the 'write enable' signal for the perpendicular-recording mode. WGATE should be set 0 for normal operation.

Figure 16 (Part 2 of 2). Command Symbols

The following figure shows the head-load time selected as determined by the HLT parameter and the transfer rate. The transfer rates are shown in 1000 of bytes per second.

HLT Parameter	Head Load Time/Transfer Rate			
	1000 kbps	500 kbps	300 kbps	250 kbps
00	128	256	426	512
01	1	2	3.3	4
02	2	4	6.7	8
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
7E	126	252	420	504
7F	127	254	423	508
Note: Delay time in milliseconds				

Figure 17. HLT Parameter Definitions

The following figure shows the head-unload-time selected as determined by the HUT parameter and the transfer rate.

HUT Parameter	Head Unload Time/Transfer Rate			
	1000 kbps	500 kbps	300 kbps	250 kbps
0	128	256	426	512
1	8	16	26.7	32
2	16	32	53.4	64
3	24	48	80.1	96
4	32	64	107	128
5	40	80	134	160
6	48	96	160	192
7	56	112	187	224
8	64	128	214	256
9	72	144	240	288
A	80	160	267	320
B	88	176	294	352
C	96	192	320	384
D	104	208	347	416
E	112	224	373	448
F	120	240	400	480
Note: Delay time in milliseconds				

Figure 18. HUT Parameter Definitions

The following figure shows the stepping rate selected as determined by the SRT parameter and the transfer rate.

SRT Parameter	Steeping Rate/Transfer Rate			
	1000 kbps	500 kbps	300 kbps	250 kbps
0	8.0	16.0	26.7	32.0
1	7.5	15.0	25.0	30.0
2	7.0	14.0	23.3	28.0
3	6.5	13.0	21.7	26.0
4	6.0	12.0	20.0	24.0
5	5.5	11.0	18.3	22.0
6	5.0	10.0	16.7	20.0
7	4.5	9.0	15.0	18.0
8	4.0	8.0	13.3	16.0
9	3.5	7.0	11.7	14.0
A	3.0	6.0	10.0	12.0
B	2.5	5.0	8.33	10.0
C	2.0	4.0	6.67	8.0
D	1.5	3.0	5.00	6.0
E	1.0	2.0	3.33	4.0
F	0.5	1.0	1.67	2.0
Note: Delay time in milliseconds				

Figure 19. SRT Parameter Definitions

The following commands are issued to the controller by the system.

Configure Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	0	0	0	1	0	0	1	1
Byte 1	0	0	0	0	0	0	0	0
Byte 2	0	EIS	EFF	PD	Threshold (THR)			
Byte 3	Precompensation Track Number (PTN)							

Figure 20. Configure Command

Result Phase: This command has no result phase.

Dumpreg Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	1	1	1	0

Figure 21. Dumpreg Command

Result Phase

	7	6	5	4	3	2	1	0
Byte 0	Present Track Number – Drive 0							
Byte 1	Present Track Number – Drive 1							
Byte 2	Present Track Number – Drive 2							
Byte 3	Present Track Number – Drive 3							
Byte 4	Stepping Rate Time				Head Unload Time			
Byte 5	Head Load Time							ND
Byte 6	Number of Sectors per Track/End of Track							
Byte 7	X	0	Reserved			X	GAP	WGATE
Byte 8	0	EIS	EFF	PD	Threshold			
Byte 9	Precompensation Track Number							

Figure 22. Dumpreg Result

Format Track Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	0	MFM	0	0	1	1	0	1
Byte 1	0	0	0	0	0	HD	US	US
Byte 2	Number of Data Bytes in Sector							
Byte 3	Sectors per Track							
Byte 4	Gap Length							
Byte 5	Fill Byte							

Figure 23. Format Track Command

Result Phase

Byte 0	Status Register 0
Byte 1	Status Register 1
Byte 2	Status Register 2
Byte 3	Reserved
Byte 4	Reserved
Byte 5	Reserved
Byte 6	Reserved

Figure 24. Format Track Result

Lock Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	Lock	0	0	1	0	1	0	0

Figure 25. Lock Command

Result Phase

Byte 0	0	0	0	Lock	0	0	0	0

Figure 26. Lock Result

Perpendicular Mode Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	0	0	1	0	0	0	1	0
Byte 1	OW	0	D3	D2	D1	D0	GAP	WGATE

Figure 27. Perpendicular Mode Command

Read Data Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	MT	MFM	SK	0	0	1	1	0
Byte 1	0	0	0	0	0	HD	US	US
Byte 2	Track Number							
Byte 3	Head Address							
Byte 4	Sector Number							
Byte 5	Number of Data Bytes in Sector							
Byte 6	End-of-Track							
Byte 7	Gap Length							
Byte 8	Data Length							

Figure 28. Read Data Command

Result Phase

Byte 0	Status Register 0
Byte 1	Status Register 1
Byte 2	Status Register 2
Byte 3	Track Number
Byte 4	Head Address
Byte 5	Sector Number
Byte 6	Number of Data Bytes in Sector

Figure 29. Read Data Result

Read Deleted Data Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	MT	MFM	SK	0	1	1	0	0
Byte 1	0	0	0	0	0	HD	US	US
Byte 2	Track Number							
Byte 3	Head Address							
Byte 4	Sector Number							
Byte 5	Number of Data Bytes in Sector							
Byte 6	End-of-Track							
Byte 7	Gap Length							
Byte 8	Data Length							

Figure 30. Read Deleted Data Command

Result Phase

Byte 0	Status Register 0
Byte 1	Status Register 1
Byte 2	Status Register 2
Byte 3	Track Number
Byte 4	Head Address
Byte 5	Sector Number
Byte 6	Number of Data Bytes in Sector

Figure 31. Read Deleted Data Result

Read ID Command

Command Phase

	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 0	0	MFM	0	0	1	0	1	0
Byte 1	0	0	0	0	0	HD	US	US

Figure 32. Read ID Command

Result Phase

Byte 0	Status Register 0
Byte 1	Status Register 1
Byte 2	Status Register 2
Byte 3	Track Number
Byte 4	Head Address
Byte 5	Sector Number
Byte 6	Number of Data Bytes in Sector

Figure 33. Read ID Result

Read Track Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	0	MFM	0	0	0	0	1	0
Byte 1	0	0	0	0	0	HD	US	US
Byte 2	Track Number							
Byte 3	Head Address							
Byte 4	Sector Number							
Byte 5	Number of Data Bytes in Sector							
Byte 6	End-of-Track							
Byte 7	Gap Length							
Byte 8	Data Length							

Figure 34. Read Track Command

Result Phase

Byte 0	Status Register 0
Byte 1	Status Register 1
Byte 2	Status Register 2
Byte 3	Track Number
Byte 4	Head Address
Byte 5	Sector Number
Byte 6	Number of Data Bytes in Sector

Figure 35. Read Track Result

Recalibrate Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	1	1	1
Byte 1	0	0	0	0	0	0	US	US

Figure 36. Recalibrate Command

Result Phase: This command has no result phase.

Relative Seek Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	1	DIR	0	0	1	1	1	1
Byte 1	0	0	0	0	0	HD	US	US
Byte 2	Relative Cylinder Number							

Figure 37. Relative Seek Command

Result Phase: This command has no result phase.

Scan Equal Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	MT	MFM	SK	1	0	0	0	1
Byte 1	0	0	0	0	0	HD	US	US
Byte 2	Track Number							
Byte 3	Head Address							
Byte 4	Sector Number							
Byte 5	Number of Data Bytes in Sector							
Byte 6	End-of-Track							
Byte 7	Gap Length							
Byte 8	Scan Test							

Figure 38. Scan Equal Command

Result Phase

Byte 0	Status Register 0
Byte 1	Status Register 1
Byte 2	Status Register 2
Byte 3	Track Number
Byte 4	Head Address
Byte 5	Sector Number
Byte 6	Number of Data Bytes in Sector

Figure 39. Scan Equal Result

Scan High or Equal Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	MT	MFM	SK	1	1	1	0	1
Byte 1	0	0	0	0	0	HD	US	US
Byte 2	Track Number							
Byte 3	Head Address							
Byte 4	Sector Number							
Byte 5	Number of Data Bytes in Sector							
Byte 6	End-of-Track							
Byte 7	Gap Length							
Byte 8	Scan Test							

Figure 40. Scan High or Equal Command

Result Phase

Byte 0	Status Register 0
Byte 1	Status Register 1
Byte 2	Status Register 2
Byte 3	Track Number
Byte 4	Head Address
Byte 5	Sector Number
Byte 6	Number of Data Bytes in Sector

Figure 41. Scan High or Equal Result

Scan Low or Equal Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	MT	MFM	SK	1	1	0	0	1
Byte 1	0	0	0	0	0	HD	US	US
Byte 2	Track Number							
Byte 3	Head Address							
Byte 4	Sector Number							
Byte 5	Number of Data Bytes in Sector							
Byte 6	End-of-Track							
Byte 7	Gap Length							
Byte 8	Scan Test							

Figure 42. Scan Low or Equal Command

Result Phase

Byte 0	Status Register 0
Byte 1	Status Register 1
Byte 2	Status Register 2
Byte 3	Track Number
Byte 4	Head Address
Byte 5	Sector Number
Byte 6	Number of Data Bytes in Sector

Figure 43. Scan Low or Equal Result

Seek Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	1	1	1	1
Byte 1	0	0	0	0	0	HD	US	US
Byte 2	New Track Number after Seek							

Figure 44. Seek Command

Result Phase: This command has no result phase.

Sense Drive Status Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	1	0	0
Byte 1	0	0	0	0	0	HD	US	US

Figure 45. Sense Drive Status Command

Result Phase

Byte 0	Status Register 3

Figure 46. Sense Drive Status Result

Sense Interrupt Status Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	1	0	0	0

Figure 47. Sense Interrupt Status Command

Result Phase

Byte 0	Status Register 0
Byte 1	Present Track Number

Figure 48. Sense Interrupt Status Result

Specify Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	0	1	1
Byte 1	Stepping Rate Time				Head Unload Time			
Byte 2	Head Load Time							ND

Figure 49. Specify Command

Result Phase: This command has no result phase.

Verify Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	MT	MFM	SK	1	0	1	1	0
Byte 1	EC	0	0	0	0	HD	US	US
Byte 2	Cylinder Address							
Byte 3	Head Address							
Byte 4	Sector Address							
Byte 5	Sector size							
Byte 6	End-of-Track							
Byte 7	Gap Length							
Byte 8	Byte Transfer Control							

Figure 50. Verify Command

Result Phase

Byte 0	Status Register 0
Byte 1	Status Register 1
Byte 2	Status Register 2
Byte 3	Cylinder Address
Byte 4	Head Address
Byte 5	Sector Address
Byte 6	Sector size

Figure 51. Verify Result

Version Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	0	0	0	1	0	0	0	0

Figure 52. Version Command

Result Phase

Byte 0	1	0	0	X	0	0	0	0

Note – x can be a 1 or a 0.

Figure 53. Version Result

Write Data Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	MT	MFM	0	0	0	1	0	1
Byte 1	0	0	0	0	0	HD	US	US
Byte 2	Track Number							
Byte 3	Head Address							
Byte 4	Sector Number							
Byte 5	Number of Data Bytes in Sector							
Byte 6	End-of-Track							
Byte 7	Gap Length							
Byte 8	Data Length							

Figure 54. Write Data Command

Result Phase

Byte 0	Status Register 0
Byte 1	Status Register 1
Byte 2	Status Register 2
Byte 3	Track Number
Byte 4	Head Address
Byte 5	Sector Number
Byte 6	Number of Data Bytes in Sector

Figure 55. Write Data Result

Write Deleted Data Command

Command Phase

	7	6	5	4	3	2	1	0
Byte 0	MT	MFM	0	0	1	0	0	1
Byte 1	0	0	0	0	0	HD	US	US
Byte 2	Track Number							
Byte 3	Head Address							
Byte 4	Sector Number							
Byte 5	Number of Data Bytes in Sector							
Byte 6	End-of-Track							
Byte 7	Gap Length							
Byte 8	Data Length							

Figure 56. Write Deleted Data Command

Result Phase

Byte 0	Status Register 0
Byte 1	Status Register 1
Byte 2	Status Register 2
Byte 3	Track Number
Byte 4	Head Address
Byte 5	Sector Number
Byte 6	Number of Data Bytes in Sector

Figure 57. Write Deleted Data Result

Invalid Command Status

Result Phase: The following status byte is returned to the system microprocessor when an invalid command has been received.

Byte 0	Status Register 0 = hex 80

Figure 58. Invalid Command Result

Note: Bits 6 and 7 in Status Register 0 are used to indicate command status. When an invalid command is processed, this information is returned to the system microprocessor in the Invalid Command Status byte.

Command Status Registers

This section provides definitions for status registers 0 through 3.

Status Register 0

Bit	Function
7, 6	Interrupt Code
5	Seek End
4	Equipment Check
3	Reserved
2	Head Address
1, 0	Drive Select

Figure 59. Status Register 0

Bits 7, 6 These bits indicate the command interrupt status.

Bits 7 6	Function
0 0	Normal Termination of Command
0 1	Abnormal Termination of Command
1 0	Invalid Command Issued
1 1	Reserved

Figure 60. Status Register 0 (Bits 7, 6)

Bit 5 This bit is set to 1 when the diskette drive completes the Seek or Recalibrate command, or a read or write operation with an implied Seek command.

Bit 4 This bit is set to 1 if the '-track 0' signal fails to occur after the Recalibrate command is issued or Relative Seek command to step outward beyond track 0.

Bit 3 Reserved. This bit is always set to 0.

Bit 2 This bit indicates the state of the '-head select' signal after the command was performed. When set to 1, head 1 was selected; when set to 0, head 0 was selected.

Bit 1, 0 These bits indicate the drive that was selected upon command completion.

Bits 1 0	Function
0 0	Drive 0
0 1	Drive 1
1 0	Drive 2
1 1	Drive 3

Figure 61. Status Register 0 (Bits 1, 0)

Status Register 1

Bit	Function
7	End-of-Track
6	Reserved
5	Cyclic Redundancy Check (CRC) Error
4	Overrun/Underrun Error
3	Reserved
2	No Data
1	Not Writable
0	Missing Address Mark

Figure 62. Status Register 1

Bit 7 This bit is set to 1 when the controller tries to gain access to a sector beyond the final sector of a track.

Bit 6 Reserved. This bit is always set to 0.

Bit 5 This bit is set to 1 when a CRC error is detected in the ID or data field.

Bit 4 This bit is set to 1 if the system does not service the diskette drive controller within an adequate period of time during data transfers.

Bit 3 Reserved. This bit is always set to 0.

Bit 2 This bit is set to 1 when:

- The controller cannot find the sector specified in the ID register during the execution of a Read Data, Read Deleted Data, or Read ID or Read Track command.
- The controller cannot read the ID field without an error during the execution of a Read ID command
- The starting sector cannot be found during the execution of a Read Track command.

Bit 1 This bit is set to 1 when the '-write-protect' signal is active during a Write Data, Write Deleted Data, or Format Track command.

Bit 0 This bit is set to 1 if the controller cannot detect an address mark. When this occurs, bit 0 of Status Register 2 indicates whether the missing address mark is an ID-address mark or a data-address mark.

Status Register 2

Bit	Function
7	Reserved
6	Control Mark
5	CRC Error in Data Field
4	Wrong Track
3	Scan Equal
2	Scan Not Satisfied
1	Bad Track
0	Missing Address Mark in Data Field

Figure 63. Status Register 2

Bit 7 Reserved. This bit is always set to 0.

Bit 6 This bit is set to 1 when the controller encounters a sector that has a deleted data-address mark during a Read Data or a Read Deleted Data encounters a data address mark.

Bit 5 This bit is set to 1 if the controller detects an error in the data.

Bit 4 This bit is set to 1 when the track number on the media is different from the track number issued by the command. When this occurs, bit 2 of Status Register 1 is also set to 1.

Bit 3 This bit is set to 1 during the Scan command when the conditions for "Equal" are satisfied.

Bit 2 This bit is set to 1 during the Scan Command when the scan conditions are not satisfied.

Bit 1 This bit is set to 1 when the track number on the media is hex FF and the track number value stored in the ID register is not hex FF. When this occurs, bit 2 of Status Register 1 is also set to 1.

Bit 0 This bit is set to 1 when the controller cannot find a data-address mark. This bit is set to 0 when an ID-address mark cannot be found. Bit 0 in Status Register 0 is also set if either address mark cannot be found.

Status Register 3

Bit	Function
7	Reserved
6	Write Protect
5	Reserved
4	Track 0
3	Reserved
2	Head Address
1, 0	Drive Select

Figure 64. Status Register 3

Bit 7	Reserved. This bit is always set to 0.
Bit 6	This bit indicates the status of the '-write-protect' signal from the diskette drive. When this bit is set to 1, the '-write-protect' signal is active.
Bit 5	Reserved. This bit is always set to 1.
Bit 4	This bit indicates the status of the '-track 0' signal from the diskette drive. When this bit is set to 1, the '-track 0' signal is active.
Bit 3	Reserved. This bit is always set to 1.
Bit 2	This bit indicates the status of the '-head 1 select' signal from the diskette drive. When this bit is set to 1, the '-head 1 select' signal is active.
Bits 1, 0	These bits indicate the current selected drive.

Interface Signal Descriptions

The following section describes the interface signals to the diskette drive.

Output Signals

All output signals to the diskette drive operate between 5 V dc and ground, with the following definitions:

- The inactive level is 2.0 V dc minimum.
- The active level is 0.8 V dc maximum.

-HIGH DENSITY SELECT: When this signal is active, the 2MB mode is selected. Diskettes are formatted with 18 sectors per track and a capacity of 1.44MB. When this signal is inactive, the 1MB mode is selected. Diskettes are formatted with 9 sectors per track and a capacity of 720KB.

DATA RATE SELECT 0 - 1: These signals are driven by the system to select the data rate of devices on the diskette drive interface. These signals are controlled by the data-rate-select bits (see Figure 12 on page 9).

-DRIVE SELECT: The drive-select signal enables or disables all drive interface signals except -MOTOR ENABLE. When the drive select signal is active, the drive is enabled. When it is inactive, all controlled inputs are ignored and all drive outputs are disabled.

-MOTOR ENABLE: When this signal is made active, the spindle starts to turn. There must be a 500-millisecond minimum delay after -MOTOR ENABLE becomes active before a read, write, or seek operation is initiated (750 milliseconds for 5.25-inch drive types). When inactive, this signal causes the spindle motor to decelerate and stop.

-DIRECTION IN: When this signal is active, -STEP moves the heads toward the drive spindle. When this signal is inactive, -STEP moves the heads away from the drive spindle. This signal is stable for at least 1 microsecond before and after the trailing edge of the -STEP pulse.

Note: After a direction change, a 15-millisecond minimum delay is required before the next '-step' pulse is issued.

-STEP: A 1-microsecond active pulse of this signal causes the read/write heads to move one track. The state of -DIRECTION at the trailing edge of -STEP determines the direction of motion.

Note: A 15-millisecond seek settle time must be provided after the last step pulse occurs before a read, write, or seek operation initiated.

-WRITE DATA: A 125-nanosecond minimum pulse of this signal causes a flux reversal to occur on the media if -WRITE ENABLE is active.

-WRITE ENABLE: When active, this signal enables the write current circuits and -WRITE DATA controls the writing of information. Motor-start and head-settle times must be observed before this signal becomes active.

-HEAD 1 SELECT: When active, this signal selects the upper head; when inactive, the lower head is selected.

Input Signals

All input signals from the drive can sink 4.0 milliamperes at the active level.

- The inactive level is 3.7 V dc minimum.
- The active level is 0.4 V dc maximum.

DRIVE TYPE ID 0 - 1: These signals provide encoded drive-type information to the system when the drive is selected.

MEDIA TYPE ID 0 - 1: These signals provide encoded media-type information to the system when the drive is selected.

-INDEX: When the drive senses the index, it generates an active pulse of at least 1 millisecond on this line.

-TRACK 0: This signal is active when the read/write head is on track 0. Track 0 is determined by a sensor, not a track counter.

The drive can seek to track 0, under control of the system even if no diskette is installed. This allows system software to determine how many drives there are attached to the system. Software selects each drive and attempts to recalibrate that drive to track 0. The track 0 indication determines whether or not each drive is installed in the system.

-WRITE PROTECT: When this signal is active the drive cannot write data to the diskette.

-READ DATA: Each flux reversal detected on the media provides a 125-nanosecond minimum active pulse on this line.

-DISKETTE CHANGE: This signal is active at power-on and latched inactive when a diskette is present, the drive is selected and a step pulse occurs. This signal goes active when the diskette is removed from the drive. The presence of a diskette is determined by a sensor.

Connector

Signals and voltages are transferred between the system board and the diskette drives by a cable or printed-circuit board. The printed-circuit board provides a 2- by 20-pin card edge connector for each diskette drive, with a locator key between pins 34 and 36. The cable interface provides a 2 x 17 pin header connector to each diskette drive, with a locator key below pin 17.

The following figures show the signals and DC voltages for each diskette drive connector type:

Pin	Signal	Pin	Signal
1	Ground	2	-High Density Select
3	Reserved	4	Reserved
5	Ground	6	Reserved
7	Ground	8	-Index
9	Ground	10	Reserved
11	Ground	12	-Drive Select
13	Ground	14	Reserved
15	Ground	16	-Motor Enable
17	Ground	18	-Direction In
19	Ground	20	-Step
21	Ground	22	-Write Data
23	Ground	24	-Write Enable
25	Ground	26	-Track 0
27	Ground	28	-Write Protect
29	Ground	30	-Read Data
31	Ground	32	-Head 1 Select
33	Ground	34	-Diskette Change
35	Ground	36	Ground
37	Ground	38	+5 V dc
39	Ground	40	+12 V dc

Figure 65. Diskette Drive Connector Signal Assignments For the 40-pin Card Edge Interface

Pin	Signal	Pin	Signal
1	Signal Return	2	-High Density Select
3	+5 V dc	4	Drive Type ID 1
5	Signal Return	6	+12 V dc
7	Signal Return	8	-Index
9	Signal Return	10	Reserved
11	Signal Return	12	-Drive Select
13	Signal Return	14	Reserved
15	Signal Return	16	-Motor Enable
17	Signal Return	18	-Direction In
19	Signal Return	20	-Step
21	Signal Return	22	-Write Data
23	Signal Return	24	-Write Enable
25	Signal Return	26	-Track 0
27	Signal Return	28	-Write Protect
29	Signal Return	30	-Read Data
31	Signal Return	32	-Head 1 Select
33	Signal Return	34	-Diskette Change

Figure 66. Diskette Drive Connector Signal Assignments For the 34-pin Header Interface

Pin	Signal	Pin	Signal
1	Signal Return	2	Data Rate Select 1
3	+5 V dc	4	Drive Type ID 1
5	Signal Return	6	+12 V dc
7	Signal Return	8	-Index
9	Drive Type ID 0	10	Reserved
11	Signal Return	12	-Drive Select
13	Signal Return	14	Reserved
15	Signal Return	16	-Motor Enable
17	Media Type ID 1	18	-Direction In
19	Signal Return	20	-Step
21	Signal Return	22	-Write Data
23	Signal Return	24	-Write Enable
25	Signal Return	26	-Track 0
27	Media Type ID 0	28	-Write Protect
29	Signal Return	30	-Read Data
31	Signal Return	32	-Head 1 Select
33	Data Rate Select 0	34	-Diskette Change

Figure 67. Diskette Drive Connector Signal Assignments For the Enhanced 34-Pin Header Interface

Index

Keyboard and Auxiliary Device Controller

Notes:

Figures

Notes:

Description

Input to the keyboard and auxiliary device controllers is through two connectors at the rear of the system unit. One connector is dedicated to the keyboard, the other is available for an auxiliary device. An auxiliary device can be any type of serial input device compatible with the controller interface. The device types include:

- Mouse
- Touchpad
- Trackball.

Both Type 1 and Type 2 controllers receive the serial data, check the parity, and present the data to the system as a byte of data at data port hex 0060. The Type 2 controller provides only 7 of the 32 internal addresses available on the Type 1, and two commands are not available on the Type 2. Also, the Type 1 controller can translate keyboard scan codes, and the Type 2 cannot. (To determine which controller is present, see the description for bit 6 of the Controller Command byte on page 10.)

Note: Because the Type 1 controller supports translation and the Type 2 does not, the keyboard must provide both scan-set 1 and scan-set 2.

The controller interrupts the system when data is available or waits for polling from the system microprocessor.

Address hex 0064 is the command/status port. When the system reads port hex 0064, it receives status information from the controller. When the system writes to the port, the controller interprets the byte as a command.

Secondary circuit protection is provided on the system board for the +5 V dc line to the keyboard and auxiliary device.

Keyboard Password

Legal password characters are restricted to the 128 ASCII character set. The password can be up to seven bytes long but must be installed using the keyboard scan codes less than hex 80 (the controller does not compare keyboard scan codes greater than hex 7F). Scan code set 1 is recommended for all password operations.

While the password is enabled, the controller compares the incoming keyboard scan code against the installed password, and it discards all data from the keyboard and auxiliary device that does not match the password. After a match occurs, the controller is restored to normal operation and data is again passed to the system microprocessor.

The controller provides three commands for keyboard password operation; it does not provide a command to verify the installed password.

A4	Test Password Installed
A5	Load Password
A6	Enable Password.

The Test Password Installed command determines if a keyboard password is currently installed. The controlling program can use this command to determine if a keyboard password is loaded before enabling the password.

The Load Password command allows the system microprocessor to set a keyboard password in the controller at any time. The existing password is lost, and the new password becomes active. The keyboard password can be changed at any time and must be installed in scan-code format.

The Enable Password command places the controller in the secure mode. While in the secure mode, the controller intercepts the keyboard data stream and continuously compares it to the installed password pattern. The controller does not pass any information to the system microprocessor or accept any commands while the keyboard password is enabled. (To enable the 'address 20' signal while the password is enabled, use System Control Port A at address hex 0092.)

The controller provides four internal RAM locations to support the keyboard password: addresses hex 13, 14, 16, and 17. See "Controller Commands" on page 9 for the commands on reading and writing to these locations.

Hex Address	**Description**
13	Security on: If this byte is nonzero when the password is enabled, the controller loads this byte into the output buffer and generates a system interrupt.
14	Security off: If this byte is nonzero when the password is matched, the controller loads this byte into the output buffer and generates a system interrupt.
16 and 17	Make 1 and 2: While the password is enabled, the controller first compares the incoming scan code against these two bytes. If the incoming scan code matches one of these two bytes, the scan code is discarded before it is compared to the password. This allows the controller to ignore certain keystrokes such as that for the right and left shift keys.

Scan Code Translation

When the Type 1 controller is in the translate mode, it converts incoming codes and presents the new code at its output port. When the controller receives a hex F0, it translates the next code received and ANDs the translated value with hex 80. For example,

```
Hex 47 is translated to hex 60
Hex F0 47 is translated hex E0 (60 AND 80)

Hex 77 is translated to hex 45
Hex F0 77 is translated to hex C5 (45 AND 80).
```

The following figure shows how the input codes are translated.

Input	Output	Input	Output	Input	Output
00	FF	30	69*	60	55*
01	43	31	31	61	56*
02	41	32	30	62	77*
03	3F	33	23	63	78*
04	3D	34	22	64	79*
05	3B	35	15	65	7A*
06	3C	36	07	66	0E
07	58*	37	5E*	67	7B*
08	64*	38	6A*	68	7C*
09	44	39	72*	69	4F
0A	42	3A	32	6A	7D*
0B	40	3B	24	6B	4B
0C	3E	3C	16	6C	47
0D	0F	3D	08	6D	7E*
0E	29	3E	09	6E	7F*
0F	59*	3F	5F*	6F	6F*
10	65*	40	6B*	70	52
11	38	41	33	71	53
12	2A	42	25	72	50
13	70*	43	17	73	4C
14	1D	44	18	74	4D
15	10	45	0B	75	48
16	02	46	0A	76	01
17	5A*	47	60*	77	45
18	66*	48	6C*	78	—
19	71*	49	34	79	4E
1A	2C	4A	35	7A	51
1B	1F	4B	26	7B	4A
1C	1E	4C	27	7C	37
1D	11	4D	19	7D	49
1E	03	4E	0C	7E	46
1F	5B*	4F	61*	7F	54
20	67*	50	6D*	80	—
21	2E	51	73*	81	—
22	2D	52	28	82	—
23	20	53	74*	83	41
24	12	54	1A	84	54
25	05	55	0D		
26	04	56	62*	F0	**
27	5C*	57	6E*		
28	68*	58	3A		
29	39	59	36		
2A	2F	5A	1C		
2B	21	5B	1B		
2C	14	5C	75*		
2D	13	5D	2B		
2E	06	5E	63*		
2F	5D*	5F	76*		

* These scan codes are reserved.
** F0 is reserved for two-byte translations.

Figure 1. Keyboard Controller Translation

Controller Status Register

The following table shows the Controller Status register.

Bit	Function
7	Parity Error
6	General Time-Out
5	Auxiliary Device Output Buffer Full
4	Inhibit Switch
3	Command/Data
2	System Flag
1	Input Buffer Full
0	Output Buffer Full

Figure 2. Controller Status Register, Read Port Hex 0064

Note: On the Type 1 controller, commands C1 and C2 place data in bits 7 through 4 of the Controller Status register. See commands C1 and C2 on page 12 for more information.

Bit 7 When set to 0, this bit indicates that the last byte of data received from the keyboard had odd parity. When a parity error occurs, this bit is set to 1 and hex FF is placed in the output buffer (the keyboard and auxiliary device use odd parity).

Bit 6 When set to 1, this bit indicates that a transmission was started by the keyboard but did not finish within the receive time-out delay, or that a transmission was started by the controller but the byte transmitted was not clocked out within the specified time limit.

When a receive time-out occurs, the controller places a hex FF in the output buffer. When a transmit time-out occurs, the controller places a hex FE in the output buffer.

The Type 1 controller also indicates a transmit time-out if a transmission was started and:

- The byte was clocked out, but a response was not received within the time limit (only this bit is set to 1).
- The byte was clocked out, but a response indicates a parity error (this bit and bit 7 are both set to 1).

Bit 5	This bit works in conjunction with bit 0. When this bit and bit 0 are set to 1, auxiliary device data is in the output buffer. When this bit is set to 0 and bit 0 is set to 1, keyboard or command controller response data is in the output buffer.
Bit 4	When set to 0, this bit indicates the password state is active and the keyboard is inhibited. When set to 1, this bit indicates the password state is inactive and the keyboard is not inhibited. See "Keyboard Password" on page 2 for more information.
Bit 3	The keyboard controller input buffer can be addressed as either address hex 0060 or 0064. Address hex 0060 is defined as the data port, and address hex 0064 is defined as the command/status port. Writing to address hex 0064 sets this bit to 1. The controller uses this bit to determine if the byte in its input buffer should be interpreted as a command byte or a data byte.
Bit 2	This bit is set to 0 or 1 by writing to the system flag bit (bit 2) in the Controller Command byte. This bit is set to 0 after a power-on reset.
Bit 1	When set to 1, this bit indicates that data has been written into the buffer, but the controller has not read the data. When the controller reads the input buffer, this bit returns to 0, indicating that the input buffer is empty. On the Type 2 controller, this bit is also set to 1 while transmitting to the keyboard or auxiliary device. After the last bit is sent, this bit is reset to 0.
Bit 0	When set to 1, this bit indicates the controller has placed data into its output buffer but the system microprocessor has not yet read the data. When the system microprocessor reads the output buffer (address hex 0060), this bit returns to 0.

Input and Output Buffers

The output buffer is an 8-bit, read-only register at address hex 0060. When the output buffer is read, the controller sends information to the system microprocessor. The information can be keyboard scan codes, auxiliary device data, or data bytes from a controller command.

Note: Do not read the output port (address hex 0060) unless the output-buffer-full bit in the Controller Status register is 1.

The input buffer is an 8-bit, write-only register at address hex 0060 or address hex 0064. When the input buffer is written to, the input-buffer-full bit (bit 1) in the Controller Status Byte is set to 1. Data written to the input buffer through address hex 0064 is interpreted as a controller command. Data written to address hex 0060 is sent to the keyboard, unless the controller expects a data byte following a controller command. Bit 3 of the Controller Status register indicates whether the contents of the input buffer is a command or a data byte.

Note: Do not write to the input port (address hex 0060) unless the input-buffer-full bit in the Controller Status register is 0.

Input and Output Ports

The input port consists of two signals driven to the controller by the keyboard and auxiliary device. The output port consists of eight signals driven by the controller to the keyboard, auxiliary device, or system interface. The following tables show the input port and the output port bytes.

Bit	Function
7−2	Reserved
1	Auxiliary Data In
0	Keyboard Data In

Figure 3. Input Port Definitions

Bit 7 - 2 Reserved.

Bit 1 This bit reflects the state of the 'data' line driven by the auxiliary device. For more information on the auxiliary device 'data' line, see "Auxiliary Device and System Timings" on page 15.

Bit 0 This bit reflects the state of the 'data' line driven by the keyboard.

Bit	Function
7	Keyboard Data Out
6	Keyboard Clock Out
5	IRQ12
4	IRQ01
3	Auxiliary Clock Out
2	Auxiliary Data Out
1	Gate Address Line 20
0	Reset Microprocessor

Figure 4. Output Port Definitions

Bit 7 This bit reflects the state of the 'data' line driven by the controller to the keyboard.

Bit 6 This bit reflects the state of the 'clock' line driven by the controller to the keyboard.

Bit 5 When set to 1, this bit indicates an interrupt has been generated by data from the auxiliary device in the output buffer. When the system reads the data from address hex 0060, this bit will be set to 0.

Bit 4 When set to 1, this bit indicates an interrupt has been generated by data from the keyboard or a command in the output buffer. When the system reads the data from address hex 0060, this bit will be set to 0.

Bit 3 This bit reflects the state of the 'clock' line driven by the controller to the auxiliary device.

Bit 2 This bit reflects the state of the 'data' line driven by the controller to the auxiliary device.

Bit 1 When this bit and bit 1 in port hex 0092 are set to 0, the system address line A20 is disabled and set to 0. This bit is set to 1 at power-on. (See "System Control Port A" in the system-specific technical reference.)

Bit 0 When set to 0, this bit resets the system microprocessor and holds it reset until the bit is set to 1.

Controller Commands

A command is a data byte written to the controller through address hex 0064. The commands are listed in order of their hex values; commands not listed are reserved.

20 – 3F Read Controller RAM: This command causes the controller to return the data contained in the internal address specified by bits 5 through 0 of this command. Internal address hex 00 is assigned as the Controller Command byte. The Type 2 controller supports only the internal addresses hex 00, 13, 14, 16, 17, 1D, and 1F.

Command hex 20 requests a read operation of the Controller Command byte. The controller returns the data to port hex 0060.

Bit	Function
7	Reserved
6	Keyboard Translate
5	Disable Auxiliary Device
4	Disable Keyboard
3	Reserved
2	System Flag
1	Enable Auxiliary Interrupt
0	Enable Keyboard Interrupt

Figure 5. Controller Command Byte

Bit 7 This bit is reserved.

Bit 6 When this bit is set to 1, the Type 1 controller translates the incoming keyboard scan codes to scan set 1. When this bit is set to 0, the controller passes the incoming scan codes without translation. Following power-on or a keyboard reset, the keyboard transmits using scan code set 2.

On the Type 2 controller, this bit cannot be set to 1; therefore, it can be used to determine the type of controller. Writing this bit as a 1 and reading it as a 0 indicates a Type 2 controller.

For keyboard operations that are compatible with IBM Personal Computers, the Type 1 controller is placed in the translate mode. To perform the same operations with the Type 2 controller, the keyboard is set up to transmit in scan code set 1 by using the Select Alternate Scan Codes command (see the Keyboard section for more information).

Note: For Type 1 controllers, this bit must be set to 0 while requesting the keyboard for its scan set. This prevents the controller from translating the keyboard response.

Bit 5	Setting this bit to 1 disables the auxiliary device interface by driving the 'clock' line low. Data is not received while the interface is disabled.
Bit 4	Setting this bit to 1 disables the keyboard interface by driving the 'clock' line low. Data is not received while the interface is disabled.
Bit 3	This bit is reserved.
Bit 2	The value written to this bit is placed in the system flag bit of the Controller Status register.
Bit 1	Setting this bit to 1 causes the controller to generate an interrupt (IRQ 12) when it places auxiliary device data into its output buffer.
Bit 0	Setting this bit to 1 causes the controller to generate an interrupt (IRQ 1) when it places keyboard or command controller response data into its output buffer.

60 - 7F Write to Controller RAM: Bits 5 through 0 of the command specify the address. Internal address hex 00 is assigned as the Controller Command byte. The Type 2 controller supports only the internal addresses hex 00, 13, 14, 16, 17, 1D, and 1F.

Warning: On the Type 2 controller, writing to unsupported internal addresses can cause the data to be transmitted to the keyboard.

Command hex 0060 writes the Controller Command byte. The next byte of data written to address hex 0060 is placed in the Controller Command byte.

A4 Test Password Installed: This command checks for a password currently installed in the controller. The test result is placed in the output buffer (address hex 0060 and IRQ 01). Hex FA means the password is installed; hex F1 means it is not installed.

A5 Load Password: This command initiates the Password Load procedure. Following this command, the controller takes input from the data port until a null (0) is detected. The null terminates password entry. (See "Keyboard Password" on page 2.)

A6 Enable Password: This command enables the controller password feature. This command is valid only when a password pattern is currently loaded in the controller. (See "Keyboard Password" on page 2.)

A7 Disable Auxiliary Device Interface: This command sets bit 5 of the Controller Command byte to 1. This disables the auxiliary device interface by driving the 'clock' line low. Data is not received while the interface is disabled.

A8 Enable Auxiliary Device Interface: This command sets bit 5 of the Controller Command byte to 0, releasing the auxiliary device interface.

A9 Auxiliary Device Interface Test: This command causes the Type 1 controller to test the auxiliary device 'clock' and 'data' lines. The following are the test results returned in the output buffer. (This command is not supported in the Type 2 controller.)

Test Result	Description
00	No error detected
01	The 'clock' line is stuck low.
02	The 'clock' line is stuck high.
03	The 'data' line is stuck low.
04	The 'data' line is stuck high.

Figure 6. Auxiliary Device Interface Test

AD Disable Keyboard Interface: This command sets bit 4 of the Controller Command byte to 1. This disables the keyboard interface by driving the 'clock' line low. Data is not received while the interface is disabled.

AE Enable Keyboard Interface: This command clears bit 4 of the Controller Command byte to 0, releasing the keyboard interface.

C0 Read Input Port: This command causes the controller to read its input port and place the data in its output buffer.

C1 Poll Input Port Low: This command is not supported by either the Type 1 or Type 2 controllers. See command C3.

C2 Poll Input Port High: This command causes the Type 1 controller to read its input port bits 7 through 4 and place them in bits 7 through 4 of the Controller Status register. This command is not supported by the Type 2 controller.

C3 Poll Input Port Low: This command causes the Type 1 controller to read its input port bits 3 through 0 and place them in bits 7 through 4 of the Controller Status register. This command is not supported by the Type 2 controller.

D0	Read Output Port: This command causes the controller to read its output port and place the data in its output buffer. This command should be used only if the output buffer is empty.
D1	Write Output Port: The next byte of data written to address hex 0060 is placed in the controller output port. For this command, the Type 2 controller supports writing to only bit 1 (gate A20). **Note:** For Type 1 controllers, bit 0 of the output port is connected to the 'reset' signal of the system microprocessor. Pulsing this bit resets the system.
D2	Write Keyboard Output Buffer: The next byte written to address hex 0060 input buffer is written to address hex 0060 output buffer as if initiated by the keyboard. An interrupt occurs if the interrupt is enabled.
D3	Write Auxiliary Device Output Buffer: The next byte written to address hex 0060 input buffer is written to address hex 0060 output buffer as if initiated by an auxiliary device. An interrupt occurs if the interrupt is enabled.
D4	Write to Auxiliary Device: The next byte written to address hex 0060 input buffer is transmitted to the auxiliary device.
| **E0**	Read Test Inputs: This command causes the Type 2 controller to read its test inputs and place the results in the output buffer. Test 0 (T0) is connected to the keyboard 'clock' line, and test 1 (T1) is connected to the auxiliary device 'clock' line. Data bit 0 represents T0, and data bit 1 represents T1. **Note:** For the Type 1 controller, these two bits are always returned as 0.
F0 - FF	Pulse Output Port: This command pulses selected bits in the controller output port for approximately 6 microseconds. Bits 3 through 0 select the respective bits in the controller output port. For example, when bit 0 of this command is set to 0, bit 0 of the output port is pulsed and the system microprocessor is reset. **Note:** The only command supported for the Type 2 controller is hex FE, pulse bit 0.

Keyboard and Auxiliary Device Programming Considerations

The following are some programming considerations for the keyboard and auxiliary device controller:

- The Controller Status register (hex 0064) can be read at any time.
- The output port (address hex 0060) should not be read unless the output-buffer-full bit in the Controller Status register is 1.
- The auxiliary-device-output-buffer-full bit in the Controller Status register indicates that the data in address hex 0060 came from the auxiliary device.
- Address hex 0060 and hex 0064 should be written to only when the input-buffer-full bit and output-buffer-full bit in the Controller Status register are 0.
- To ensure that the buffer data is valid, disable the keyboard and auxiliary devices before initiating a command that causes the controller to generate output at port 60 (such as commands D1 and D3).
- When polling the Type 1 controller for the output-buffer-full condition, wait 7 microseconds from the buffer-full indication in the Controller Status register before reading the output buffer.

Auxiliary Device and System Timings

Data transmissions to and from the auxiliary device connector consist of an 11-bit data stream sent serially over the 'data' line. The following table shows the function of each bit.

Bit	Function
11	Stop bit (always 1)
10	Parity Bit (odd parity)
9	Data Bit 7 (most-significant)
8	Data Bit 6
7	Data Bit 5
6	Data Bit 4
5	Data Bit 3
4	Data Bit 2
3	Data Bit 1
2	Data Bit 0 (least-significant)
1	Start Bit (always 0)

Figure 7. Bit Definitions of Auxiliary-Device Data Stream

The parity bit is either 1 or 0, and the 8 data bits, plus the parity bit, always have an odd number of 1's.

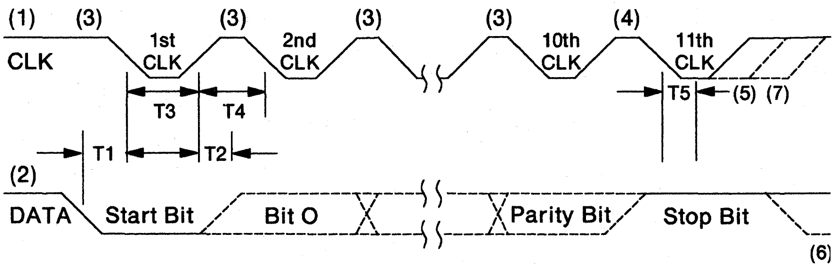
System Receiving Data

The following describes the typical sequence of events when the system is receiving data from the auxiliary device.

1. The auxiliary device checks the 'clock' line. If the line is inactive, output from the device is not allowed.
2. The auxiliary device checks the 'data' line. If the line is inactive, the controller receives data from the system.
3. The auxiliary device checks the 'clock' line during the transmission at intervals not exceeding 100 microseconds. If the device finds the system holding the 'clock' line inactive, the transmission is terminated. The system can terminate transmission anytime during the first 10 clock cycles.
4. A final check for terminated transmission is performed at least 5 microseconds after the 10th clock.
5. The system can hold the 'clock' signal inactive to inhibit the next transmission.
6. The system can set the 'data' line inactive if it has a byte to transmit to the device. The 'data' line is set inactive when the start bit (always 0) is placed on the 'data' line.

7. The system raises the 'clock' line to allow the next transmission.

The following are the timings for data received from the auxiliary device.



	Timing Parameter	Min/Max
T1	Time from DATA transition to falling edge of CLK	5/25 µs
T2	Time from rising edge of CLK to DATA transition	5/T4 - 5 µs
T3	Duration of CLK inactive	30/50 µs
T4	Duration of CLK active	30/50 µs
T5	Time to auxiliary device inhibit after clock 11 to ensure the auxiliary device does not start another transmission	>0/50 µs

Figure 8. Receiving Data Timings

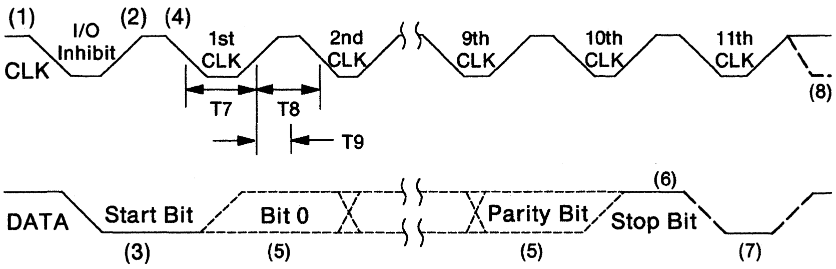
System Sending Data

The following describes the typical sequence of events when the system is sending data to the auxiliary device.

1. The system checks for an auxiliary device transmission in process. If a transmission is in process and beyond the 10th clock, the system must receive the data.
2. The auxiliary device checks the 'clock' line. If the line is inactive, an I/O operation is not allowed.
3. The auxiliary device checks the 'data' line. If the line is inactive, the system has data to transmit. The 'data' line is set inactive when the start bit (always 0) is placed on the 'data' line.
4. The auxiliary device sets the 'clock' line inactive. The system then places the first bit on the 'data' line. Each time the auxiliary device sets the 'clock' line inactive, the system places the next bit on the 'data' line until all bits are transmitted.
5. The auxiliary device samples the 'data' line for each bit while the 'clock' line is active. Data must be stable within 1 microsecond after the rising edge of the 'clock' line.

6. The auxiliary device checks for a positive-level stop bit after the 10th clock. If the 'data' line is inactive, the auxiliary device continues to clock until the 'data' line becomes active. Then it clocks the line-control bit and, at the next opportunity, sends a Resend command to the system.
7. The auxiliary device pulls the 'data' line inactive, producing the line-control bit.
8. The system can pull the 'clock' line inactive, inhibiting the auxiliary device.

| The following are the timings for data sent to the auxiliary device.



	Timing Parameter	Min/Max
T7	Duration of CLK inactive	30/50 μs
T8	Duration of CLK active	30/50 μs
T9	Time from inactive to active CLK transition, used to time when the auxiliary device samples DATA	5/25 μs

Figure 9. Sending Data Timings

Signals

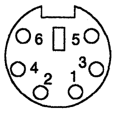
The keyboard and auxiliary device signals are driven by open-collector drivers pulled to 5 V dc through a pull-up resistor. The following lists the characteristics of the signals.

Sink Current	20 mA	Maximum
High-Level Output Voltage	5.0 V dc minus pullup	Minimum
Low-Level Output Voltage	0.5 V dc	Maximum
High-Level Input Voltage	2.0 V dc	Minimum
Low-Level Input Voltage	0.8 V dc	Maximum

Figure 10. Keyboard and Auxiliary Device Signals

Connector

The keyboard and auxiliary device connectors use 6-pin miniature DIN connectors. The signals and voltages are the same for both connectors and are assigned as shown in the following table.



Pin	I/O	Signal Name
1	I/O	Data
2	NA	Reserved
3	NA	Ground
4	NA	+5 V dc
5	I/O	Clock
6	NA	Reserved

Figure 11. Keyboard and Auxiliary Device Connector Information

Serial Port Controller

Figures

Notes:

Description

The serial port controller is programmable and supports asynchronous communications. The controller automatically adds and removes start, stop, and parity bits. A programmable baud-rate generator allows operation from 50 baud to 345.6KB. The controller supports 5-, 6-, 7-, and 8-bit characters with 1, 1.5, or 2 stop bits. A prioritized interrupt system controls transmit, receive, error, line status, and data-set interrupts.

The serial port controller provides the following functions:

- Full double buffering in the character mode, eliminating the need for precise synchronization
- False-start bit detection
- Line-break generation and detection
- Modem control functions:
  - Clear to send (CTS)
  - Request to send (RTS)
  - Data set ready (DSR)
  - Data terminal ready (DTR)
  - Ring indicator (RI)
  - Data carrier detect (DCD).

| Four types of serial port controllers have been used on the system boards.

- To programs, the Type 1 controller appears to be identical to the serial port on the IBM Personal Computer AT IBM Personal Computer Serial/Parallel Adapter.
- The Type 2 controller incorporates all functions of the Type 1 and also provides support of the first-in-first-out (FIFO) mode.
- The Type 3 controller incorporates all functions of the Type 2 controller and provides the Direct Memory Access (DMA) mode.
- | The Type 4 controller incorporates all the functions of the Type 3 controller and provides additional I/O addresses.

Note: Some systems using the Type 2 controller do not support the FIFO mode. For information about individual systems refer to the system-specific technical reference manuals.

Support for the Type 1 controller is restricted to the functions that are identical to the NS16450. Using the Type 1 controller in the FIFO mode can result in nondetectable data errors. See "Registers" on page 12 for detailed FIFO information.

The following figure is a block diagram of the serial port controller.

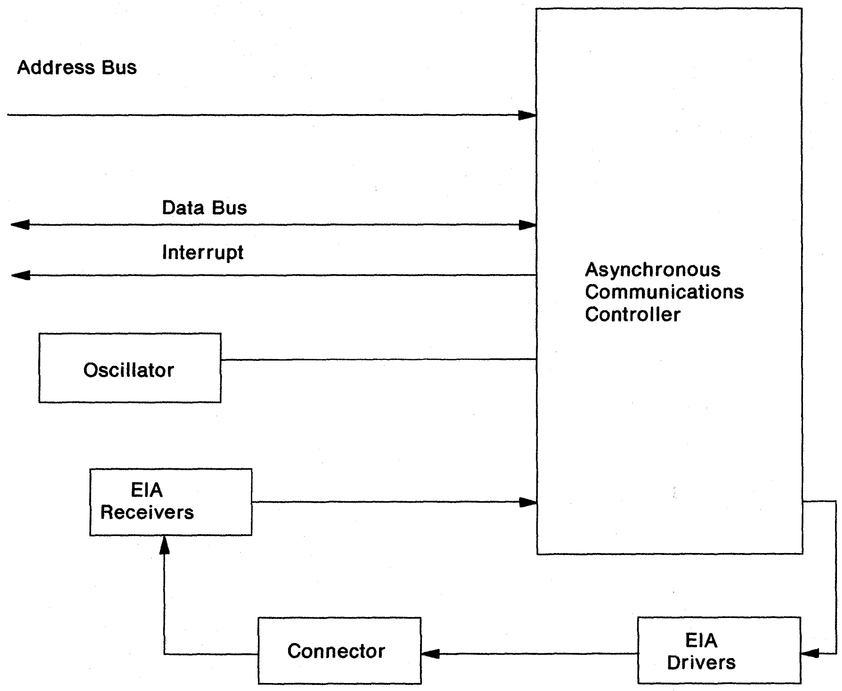


Figure 1. Serial Port Controller Block Diagram

Communications Application

For Type 1 and Type 2 controllers, the serial port can be addressed as either of two serial ports (Serial 1 or Serial 2). The Type 3 controller can be addressed as any one of eight serial ports (Serial 1 through Serial 8). The Type 4 can be addressed as any one of 16 serial ports (Serial 1 through Serial 16). The following table illustrates the base addresses and their corresponding serial ports:

Serial Port	Compatible (Hex)	Enhanced (Hex)	Types Supported
Serial 1	03F8	83F8	1,2,3,4
Serial 2	02F8	82F8	1,2,3,4
Serial 3	3220	B220	3,4
Serial 4	3228	B228	3,4
Serial 5	4220	C220	3,4
Serial 6	4228	C228	3,4
Serial 7	5220	D220	3,4
Serial 8	5228	D228	3,4
Serial 9	3A20	BA20	4
Serial 10	3A28	BA28	4
Serial 11	3A30	BA30	4
Serial 12	3A38	BA38	4
Serial 13	3A80	BA80	4
Serial 14	3A88	BA88	4
Serial 15	3A90	BA90	4
Serial 16	3A98	BA98	4

Figure 2. Serial Port Register - Base Addresses

The Type 1 and Type 2 controllers support only the compatible registers. Type 3 and Type 4 controllers support both the compatible and the enhanced registers. In this section, serial port register addresses contain a ***base c*** or a ***base e*** to signify the compatible- or enhanced-register base address, followed by an offset to be added to the base address to get the effective address of the register. The register assignments are controlled by Programmable Option Select (POS) and are made during system board setup.

Two interrupt lines are provided to the system. For Type 1 and Type 2 controllers, interrupt level 4 (IRQ4) is for Serial 1 and interrupt level 3 (IRQ3) is for Serial 2. For Type 3 and Type 4, the interrupt level assigned to the serial port is independent of the addresses. Either of the interrupt levels (IRQ3 and IRQ4) can be assigned to any of the eight serial ports. For the serial port controller to send interrupts to the interrupt controller, bit 3 of the Modem Control register must be set to 1.

The data format is shown in the following figure.

Mark-ing	Start Bit	D0	D1	D2	D3	D4	D5	D6	D7	Parity Bit	Stop Bit

Figure 3. Serial Port Data Format

Data bit 0 (D0)is the first bit to be sent or received. The controller automatically inserts the start bit, the correct parity bit (if programmed to do so), and the stop bits (1, 1.5, or 2 depending on the command in the Line Control register).

Programmable Baud-Rate Generator

The controller has a programmable baud-rate generator that can divide the clock input (11.0592 MHz) by any divisor from 1 to 65,535. In compatibility mode, a 1.8432 MHz clock is used. The output frequency of the baud-rate generator is the baud rate multiplied by 16. Two 8-bit latches store the divisor in a 16-bit binary format. The divisor latches are loaded during setup to ensure desired operation of the baud-rate generator. When either of the divisor latches is loaded, a 16-bit baud counter is immediately loaded. This prevents long counts on the first load.

Modem Status Interrupts

The modem status interrupts occur as soon as the corresponding input signals change state, whether the Received Data Status register is enabled or not. The current modem status is immediately available in the Modem Status register. However, the change in the modem status is reflected in the received-data-status character that follows the actual change.

Whenever the overrun error occurs, the interrupt is generated and the corresponding bit in the Line Status register is set. In character mode, when an overrun error occurs, the character in the Receiver Buffer register is overwritten. In the FIFO or DMA mode, when an overrun error occurs, the data in the FIFO mode is preserved, and the character in the Receive Shift register is overwritten. When the Received Data Status register is enabled, the overrun error is indicated in the received-status-byte of the first character received after the last error. The indicator stays on for only one character regardless of the number of characters lost, unless another error occurs.

The interrupts for parity error and frame error occur when the error character is the next one to be read from the FIFO mode in DMA, whether the Received Data Status register is enabled or not. Although these interrupts are reset by reading the Line Status register, the bits can cause an interrupt, but they do not identify which character had the error in DMA mode. The Received Data Status register must be enabled to determine which character had the error.

FIFO Modes of Operation

The serial port contains two register stacks of 16 bytes each. These register stacks are called FIFOs. One is the Receive FIFO and the other is the Transmit FIFO.

In the FIFO mode, the controller can operate in the interrupt mode or the polled mode. To enable the FIFO mode, set bit 0 in the FIFO Control register to 1.

Interrupt Mode

When the receiver interrupts are enabled in the Interrupt Enable register, they occur as follows:

- A received-data-available interrupt is issued to the system when the FIFO register has reached the programmed trigger level.
- The Interrupt Identification register's received-data-available condition is set when the trigger level is reached and, like the interrupt, is cleared when the register drops below the trigger level.
- The receiver-line-status interrupt has a higher priority than the received-data-available interrupt.
- Bit 0 in the Line Status register is set to 1 to indicate that a character is transferred from the Shift register to the FIFO register. It is set to 0 when the Receiver FIFO register is empty.

When the Receiver FIFO register and receiver interrupts are enabled, the following occurs:

- A FIFO time-out interrupt occurs if the following conditions exist:
  - At least 1 character is in the Receiver FIFO register.

  - The last character was received more than four continuous-character times ago (if 2 stop bits are programmed, the second one is included in this time delay).
  - The most recent system microprocessor read off the Receiver FIFO register was longer than four continuous-character times ago.

  This causes a maximum character-received to interrupt-issued delay of 160 milliseconds at 300 baud, with a 12-bit character.

- Character times are calculated by using the 'receiver clock' input for a clock signal (this makes the delay proportional to the baud rate).
- When a time-out interrupt has occurred, it is cleared, and the timer is reset when the system microprocessor reads one character from the Receiver FIFO register.
- When a time-out interrupt has not occurred, the time-out timer is reset after a new character is received, or after the system microprocessor reads the Receiver FIFO register.

When the Transmitter FIFO register and transmitter interrupts are enabled (FIFO Control register bit 0 and Interrupt Enable register bit 1 are set to 1), the following occurs:

- The transmitter-holding-register-empty interrupt (02) occurs when the Transmitter FIFO register is empty. It is cleared when the Transmitter Holding register is written to (1 to 16 characters can be written to the Transmitter FIFO register while this interrupt is being serviced), or the Interrupt Identification register is read.
- The transmitter-FIFO-register-empty indications are delayed one character time minus the last stop-bit time whenever both of the following occur:
  - Bit 5 (transmitter-holding-register-empty) of the Line Status register is set to 1.
  - There have not been at least two bytes in the Transmitter FIFO register at the same time since the last time bit 5 of the Line Status register was set to 1.

  The first transmitter interrupt after changing bit 0 in the FIFO Control register is immediate, if enabled.

Character time-out and Receiver FIFO register trigger-level interrupts have the same priority as the current received-data-available interrupt. The transmitter-FIFO-register-empty interrupt has the same priority as the current transmitter-holding-register-empty interrupt.

Polled Mode

To put the controller in the FIFO polled mode, disable the interrupts through the Interrupt Enable register and enable the FIFO mode. The Receiver and Transmitter FIFO registers are controlled separately. Either or both registers can be in the polled mode of operation.

In the FIFO-polled mode of operation, the system reads the status of the Receiver and Transmitter FIFO register through the Line Status register.

- The data-ready bit indicates whether or not the Receiver FIFO register contains data.
- The error bits indicate the type of error. Character error status is handled the same way as when in the interrupt mode. The Interrupt Identification register is not affected because bit 2 of the Interrupt Enable register is set to 0.
- Line Status register bit 5 indicates when the Transmitter FIFO register is empty.
- Line Status register bit 6 indicates that both the Transmitter FIFO register and Transmitter Shift register are empty.
- Line Status register bit 7 indicates any errors in the Receiver FIFO register.

There is no trigger level reached or time-out condition indicated in the FIFO polled mode; however, the Receiver and Transmitter FIFO registers are still fully capable of holding characters.

DMA Modes of Operation

In addition to the character mode and the FIFO mode of the Type 2 controller, the Type 3 and Type 4 controllers support the use of DMA for receiving and transmitting. These controllers also provide new functions and new interrupts, many of which are available in the FIFO mode.

| The presence of the Type 3 or Type 4 controller can be detected by
| enabling the DMA transmit mode (bit 6 in Enhanced Function register 1), reading bits 6 and 7 of the Interrupt ID register, and then disabling the DMA transmit mode. If bit 6 is a 1 and bit 7 is a 0, a Type 3 or Type 4 controller is installed. If the FIFO mode is enabled, but not the DMA transmit mode, then bits 6 and 7 in the Interrupt ID register read as 1. If the character mode is enabled, then bits 6 and 7 are 0.

The DMA mode uses separate DMA channels for transmitting and receiving. In addition, the transmit and receive modes may be set independently. (Operating the receiver in DMA mode and the transmitter in FIFO mode will conserve DMA channels). This allows a high performance receiving function and requires only one interrupt per 16 characters transmitted.

Receive Mode

| While in receive mode, the controller signals a request to transfer data when the Receive FIFO register has reached the receiver trigger level or when a timeout occurs. The data is then transferred from the controller until the FIFO register is empty or the DMA Terminal Count (TC) is reached. A timeout occurs if there is at least one character in the FIFO register and a character has not been read in the last four character times. An interrupt on the transmit terminal count and the receive terminal count occur independently.

Transmit Mode

While in the transmit mode, data is transferred until the FIFO register is full or the end of the data is reached.

Two separate terminal count interrupts are available: one for transmit and the other for receive.

Received Data Status Register

When received, a status byte can be placed in the FIFO register with each data byte. This option is available in the DMA receive mode and in the FIFO mode. The status byte is defined as the Received Data Status register and is stored after the corresponding data byte. The bit definitions of Received Data Status register are as follows:

Bit	Description
7	Data Carrier Detect
6	Clear to Send
5	Data Set Ready
4	Break
3	Framing
2	Parity Error
1	Overrun Error
0	Error/Break

Figure 4. Received Data Status Register

Bit 7	This bit indicates the state of the '-data carrier detect' signal (-DCD).
Bit 6	This bit indicates the state of the '-clear to send' signal (-CTS).
Bit 5	This bit indicates the state of the '-data send ready' signal (-DSR).
Bit 4	This bit determines Break (BI)
Bit 3	This bit determines Framing (FE)
Bit 2	This bit determines Parity Error (PE)
Bit 1	This bit determines Overrun Error (OE)
Bit 0	This bit determines the Error/Break. Bit 0 is set and reset.

Bit 0 is set to 1 when an error or break occurs and remains set for subsequent characters until it is reset to 0 by a software command (write hex 03 to the Enhanced Command register). This allows scanning of the received data (in memory or as it exits from the FIFO mode) to find the character that was received with an error. While the error/break bit indicates that an error has occurred, it is not possible to distinguish multiple errors from single errors without checking the status of each character that has this bit set.

Bit 0 is set to 1 and reset to 0 before the Received Data status register is placed in the FIFO mode. This means that up to eight characters have bit 0 set after the Reset Error/Break Indicator command is given and there are no additional errors.

When the Received Data Status register is enabled the Receive FIFO register can hold eight data bytes and eight status bytes.

Transmit Commands

New transmit commands provide better software control of the transmitter and allow the insertion of special control characters such as XON and XOFF into the the transmitted data stream. The new transmit commands are available in both the DMA transmit mode and in the FIFO mode. The new transmit commands are:

- Start Sending - starts or continues transmit
- Stop Sending - stops transmit.

To insert a character in the transmitted data stream, stop the transmitter by issuing the Stop Sending command, write a character to the Transmitter Holding register, and then start the transmitter by issuing the Start Sending command.

Modem Pacing

Modem pacing is handled by several new functions without software involvement, which are available in the DMA mode and the FIFO mode. These new functions also prevent the async port from receiving invalid data.

The transmitter and receiver are controlled by the following signals:

- The '-clear-to-send' signal
  - When this signal is equal to 0, the transmitter is turned off.
- The '-data-carrier-detect' signal
  - When this signal is equal to 0, the transmitter is turned off.
- The '-data-set-ready' signal
  - When this signal is equal to 0, the transmitter and the receiver are turned off.

Character Orientated Pacing

Three software-programmable registers are provided to handle character oriented pacing. The contents of these registers are compared to each received character. If there is a match, a preprogrammed action takes place. The possible actions on a match are interrupt, delete character, stop transmitter, and start transmitter. If an error or break occurs in a received character, it is not compared.

Receive Character Count Register

This register enables the user to keep track of the number of characters sent to the central processing unit or the DMA. The Receive Character Count Interrupt is asserted when the counter is decremented to 0, provided that bit 0 of the Enhanced Function register 1 is set.

Byte Pacing

While in the DMA mode, the Byte Pacing function is useful when the controller is communicating with a slow processor. This function allows the controller to transmit every byte at 16 times the Receive Character Count value or 256 times the RCCR value. The result of this product is called RCLK time. (See "Receive Character Count Register (Base e+7)" on page 40 for more information.)

Enhanced Interrupts

Several new interrupts are available to support the DMA mode, the Receive Character Count register, and the Character Compare registers. The new interrupts are:

- Interrupt on Transmitter FIFO and transmitter-shift-register-empty
- Interrupt on Terminal Count in the DMA transmit mode
- Interrupt on Terminal Count in the DMA receive mode
- Interrupt on Receiver Character Count equals 0
- Interrupt on Character Compare Register Match.

Serial Port Controller Programming Considerations

The serial port uses either of the four serial communications controllers. The following should be considered when programming the serial controller:

- The Type 1 serial controller does not support the FIFO mode.
- Some systems using the Type 2 controller do not support the FIFO mode. For more information, refer to the system-specific technical reference manuals.
- The system configuration utility is used to configure serial port on the system board. The Type 1 and Type 2 controller can be configured as either Serial 1 or Serial 2, the Type 3 can be configured as Serial 1 through 8, and the Type 4 can be configured as Serial 1 through Serial 16.
- Before changing the Line Control register, make sure the Transmitter Holding register is empty.

Registers

The controller has several accessible registers. These control the operations of the controller and transmit and receive data. The system programmer can gain access to or control any of the controller registers through the system microprocessor.

Compatible Registers

| The each of four types of serial port controller has certain registers
| that are common to them all. These registers will be referred to as
| Compatible registers. The Type 3 and Type 4 controllers have
| additional registers that Type 1 and Type 2 controllers do not have.
| These registers are referred to as the Enhanced registers.

| In addition to these serial port registers, the Type 4 controller has an
| additional register that is used to select the arbitration level. The
| address of this register depends on the address range selected.

Serial Port	Compatible (Hex)	Enhanced (Hex)	Arbitration
Serial 1	03F8-03FF	83F8-83FF	4620
Serial 2	02F8-02FF	82F8-82FF	4621
Serial 3	3220-3227	B220-B227	4622
Serial 4	3228-322F	B228-B22F	4623
Serial 5	4220-4227	C220-C227	4624
Serial 6	4228-422F	C228-C22F	4625
Serial 7	5220-5227	D220-D227	4626
Serial 8	5228-522F	D228-D22F	4627
Serial 9	3A20-3A27	BA20-BA27	4628
Serial 10	3A28-3A2F	BA28-BA2F	4629
Serial 11	3A30-3A37	BA30-BA37	462A
Serial 12	3A38-3A3F	BA38-BA3F	462B
Serial 13	3A80-3A87	BA80-BA87	462C
Serial 14	3A88-3A8F	BA88-BA8F	462D
Serial 15	3A90-3A97	BA90-BA97	462E
Serial 16	3A98-3A9F	BA98-BA9F	462F

Figure 5. Serial Port Register - Base Addresses

The bit definitions of the Interrupt Enable register, Interrupt Identification register, and Line Status register have been modified from the Type 1 controller registers. A FIFO Control register has been added to support the FIFO mode.

Note: Using the Type 1 controller in the FIFO mode can result in nondetectable data errors.

Specific registers are selected according to the figure below and the figure on the following page.

Address Offsets	R/W	Register
+ 0 *	W	Transmitter Holding Register
+ 0 *	R	Receiver Buffer Register
+ 0 *	R/W	Divisor Latch, Low Byte
+ 1 *	R/W	Divisor Latch, High Byte
+ 1 *	R/W	Interrupt Enable Register
+ 2	R	Interrupt Identification Register
+ 2	W	FIFO Control Register
+ 3	R/W	Line Control Register
+ 4	R/W	Modem Control Register
+ 5	R	Line Status Register
+ 6	R/W	Modem Status Register
+ 7	R/W	Scratch Register

Note: *The DLAB state is controlled by bit 7 of the Line Control register.

Figure 6. Serial Port Compatible Register Address Offsets

Port Address Offset	EFR3 Bits 2 1 0	R/W	Register
0	x x x	W	Enhanced Command
1	x x x	R	Reserved
2	x x x	R	Enhanced Interrupt Identification
3	x x x	R/W	Enhanced Function 1
4	x x x	R/W	Enhanced Function 2
5	x x x	R/W	Enhanced Function 3
5 *	0 0 0	R/W	Char Compare Function 0
5 *	0 0 1	R/W	Char Compare 0
5 *	0 1 0	R/W	Char Compare Function 1
5 *	0 1 1	R/W	Char Compare 1
5 *	1 0 0	R/W	Char Compare Function 2
5 *	1 0 1	R/W	Char Compare 2
6 *	x x x	R/W	Char Compare Data
7	x x x	R/W	Receive Character Count

*The Char Compare Function Register (CCFR) and the Char Compare Register (CCR) are selected by writing the address to the Enhanced Function Register 3 and the data is read or written by reading or writing to the Char Compare Data Register (CCDR).

Figure 7. Serial Port Enhanced Register Address Offsets

Arbitration Register (Hex 462x)

This 8-bit read/write register selects the arbitration level used for the transmit and receive operations for the Type 4 controller. This register is available on the Type 4 controller only; its address corresponds to the address range assigned for the enhanced registers. The address is hex 4620 for Serial 1 and hex 462F for Serial 16.

Bits	Description
7−4	Transmit Arbitration Level
3−0	Receive Arbitration Level

Figure 8. Arbitration Register (Hex 462x)

Transmitter Holding Register (Base c + 0)

The Transmitter Holding register contains the character to be sent when the divisor latch access bit 1 (DLAB) is 0. Bit 0 is the least-significant bit and the first bit sent serially, as shown below.

Bit	Description
7	Data Bit 7
6	Data Bit 6
5	Data Bit 5
4	Data Bit 4
3	Data Bit 3
2	Data Bit 2
1	Data Bit 1
0	Data Bit 0

Figure 9. Transmitter Holding Register (Base c + 0)

Receiver Buffer Register (Base c+0)

The Receiver Buffer register contains the received character and can be accesses when the divisor-latch-access bit (DLAB) equals 0. Bit 0 is the least-significant bit and the first bit received serially, as shown in the following figure.

Bit	Description
7	Data Bit 7
6	Data Bit 6
5	Data Bit 5
4	Data Bit 4
3	Data Bit 3
2	Data Bit 2
1	Data Bit 1
0	Data Bit 0

Figure 10. Receiver Buffer Register (Base c+0)

Divisor Latch Register (Base c+1)

The Divisor Latch register is used to program the baud-rate generator. The value in this register forms the divisor of the clock input (1.8432 MHz or 11,0592MHz), which establishes the desired baud-rate (DLAB=1).

Bit	Description
7	Bit 7
6	Bit 6
5	Bit 5
4	Bit 4
3	Bit 3
2	Bit 2
1	Bit 1
0	Bit 0

Figure 11. Divisor Latch Register, Low Byte (Base c+1)

Note: If bit 6 of the Enhanced Function register 2 is set to 0, then the input clock of the baud-rate generator is 1.8432 MHz. Otherwise, the input clock is 11.0592 MHz.

Divisor Latch Register (Base c + 0)

The Divisor Latch register is used to program the baud-rate generator. The value in this register forms the divisor of the clock input (1.8432 MHz or 11,0592MHz), which establishes the desired baud-rate (DLAB = 0).

Bit	Description
7	Bit 7
6	Bit 6
5	Bit 5
4	Bit 4
3	Bit 3
2	Bit 2
1	Bit 1
0	Bit 0

Figure 12. Divisor Latch Register, Low Byte (Base c + 0)

Bit	Description
7	Bit 15
6	Bit 14
5	Bit 13
4	Bit 12
3	Bit 11
2	Bit 10
1	Bit 9
0	Bit 8

Figure 13. Divisor Latch Register, High Byte (Base c + 1)

The following shows how the baud rate is determined with an input frequency of 1.8432 MHz.

Note: Data speed should not exceed 19,200 baud (For Type 1 and Type 2).

Desired Baud Rate Rate	Divisor Used to Generate 16x Clock		Percentage of Error Difference between Desired and Actual
	Decimal	Hex	
50	2304	900	--
75	1536	600	--
110	1047	417	0.026
134.5	857	359	0.058
150	768	300	--
300	384	180	--
600	192	C0	--
1200	96	60	--
1800	64	40	--
2000	58	3A	0.69
2400	48	30	--
3600	32	20	--
4800	24	18	--
7200	16	10	--
9600	12	C	--
19200	6	6	--

Figure 14. Baud Rates at 1.8432 MHz (Low Frequency Mode)

The following shows how the baud rate is determined with an input frequency of 11.0592 MHz.

Desired Baud Rate Rate	Divisor Used to Generate 16x Clock		Percentage of Error Difference between Desired and Actual
	Decimal	Hex	
50	13824	3600	--
75	9216	2400	--
110	6284	188C	0.006
134.5	5139	1413	0.001
150	4608	1200	--
300	2304	900	--
600	1152	480	--
1200	576	240	--
1800	384	180	--
2000	346	15A	0.116
2400	288	120	--
3600	192	C0	--
4800	144	90	--
7200	96	60	--
9600	72	48	--
19200	36	24	--
31250	22	16	0.538
38400	18	12	--
57600	12	C	--
115200	6	6	--
172800	4	4	--
345600	2	2	--
Note: Divisor of 1 not supported. Data speed must not exceed 345.6 kbaud.			

Figure 15. Baud Rates at 11.0592 MHz (High Frequency Mode)

Interrupt Enable Register (Base c + 1)

This 8-bit register allows the four types of controller interrupts to separately activate the 'chip interrupt output' signal. The interrupt system can be completely disabled by setting bits 0 through 3 of the Interrupt Enable register to 0. Similarly, by setting the appropriate bits of this register to 1, selected interrupts can be enabled. Disabling prevents the controller from generating the external interrupt to the system. All other system functions operate normally, including the setting of the Line Status and Modem Status registers (DLAB = 0).

Bit	Description
7 − 4	Reserved = 0
3	Modem-Status Interrupt
2	Receiver-Line-Status Interrupt
1	Transmitter-Holding-Register-Empty Interrupt
0	Received-Data-Available Interrupt (Character and FIFO Mode) and Time-Out Interrupts (FIFO Mode Only)

Figure 16. Interrupt Enable Register (Base c + 1)

Bits 7 − 4 These bits are reserved and always set to 0.

Bit 3 When set to 1, this bit enables the modem-status interrupt.

Bit 2 When set to 1, this bit enables the receiver-line-status interrupt.

Bit 1 When set to 1, this bit enables the transmitter-holding-register-empty interrupt.

Bit 0 When set to 1, this bit enables the received-data-available interrupt. In the FIFO mode, this bit also enables the time-out interrupts.

FIFO Control Register (Base c + 2)

The FIFO Control register is a write-only register at the same location as the read-only Interrupt Identification register. The FIFO Control register enables the FIFO registers, clears the FIFO registers, and sets the Receiver FIFO register trigger level.

Note: The Transmitter and Receiver FIFO registers are not accessible serial controller registers.

The contents of the FIFO Control register are shown in the following figure.

Bit	Description
7, 6	Receiver FIFO Register Trigger
5−3	Reserved = 0
2	Transmitter FIFO Register Reset
1	Receiver FIFO Register Reset
0	FIFO Mode Enable

Figure 17. FIFO Control Register (Base c + 2)

Bits 7, 6 These bits select the trigger level for the receiver-register interrupt, as shown in the following figure.

Bits 7 6	Receiver Trigger Level
0 0	1 Byte
0 1	4 Bytes
1 0	8 Bytes
1 1	14 Bytes

Figure 18. Trigger Level

Bits 5 − 3 These bits are reserved and always set to 0.

Bit 2 When this bit is set to 1, all bytes in the Transmitter FIFO register are cleared and its counter logic is reset to 0. The Transmitter Shift register is not cleared. This bit is self-clearing.

Bit 1 When this bit is set to 1, all bytes in the Receiver FIFO register are cleared and its counter logic is reset to 0. The Transmitter Shift register is not cleared. This bit is self-clearing.

Bit 0 When this bit is set to 1, the FIFO mode is enabled. When this bit is changed, the transmit and receive (XMIT and RCV) FIFOs are cleared. When writing to any other FIFO Control register bits, this bit must be a 1.

Interrupt Identification Register (Base c+2)

To minimize programming overhead during character mode transfers, the controller prioritizes interrupts into four levels:

- Priority 1 - Receiver-line-status
- Priority 2 - Received-data-available
- Priority 2 - Time-out (FIFO mode)
- Priority 3 - Transmitter-holding-register-empty
- Priority 4 - Modem status.

Information about a pending interrupt is stored in the Interrupt Identification register. When this register is addressed, the pending interrupt with the highest priority is held, and no other interrupts are acknowledged until the system microprocessor services that interrupt.

Bit	Description
7, 6	FIFO Registers Enabled
5, 4	Interrupt ID, Bit 4
3	Interrupt ID, Bit 2
2	Interrupt ID, Bit 1
1	Interrupt ID, Bit 0
0	Interrupt Not Pending

Figure 19. Interrupt Identification Register (Base c+2)

Bits 7,6	Programs can determine which type of controller is present by reading these two bits when bit 0 of the FIFO Control register is set to 1. If bits 7 and 6 are set to 1, the Type 2 controller is present and FIFO support is provided. If bit 6 is set to 0, the controller is a Type 1 and the FIFO mode should not be used.

Bits 7 6	**Version**
0 0	Type 1 Controller
1 0	N/A
0 1	Type 3 or Type 4 Controller
1 1	Type 2 Controller

Figure 20. Type controllers for Bits 7 and 6

Note: Some systems using the Type 2 controller do not support the FIFO mode. For information about individual systems refer to the system-specific technical reference manuals.

Bits 5, 4 These bits are always set to 0.

Bit 3 In the FIFO mode, this bit is set to 1, along with bit 2, to indicate that a time-out interrupt is pending. In the character mode, this bit is always set to 0.

Bits 2, 1 These two bits identify the pending interrupt with the highest priority.

Bit 0 When this bit is set to 1, no interrupt is pending and polling (if used) continues. When this bit is set to 0, an interrupt is pending, and the contents of this register can be used as a pointer to the appropriate interrupt service routine. This bit can be used in hard-wired, prioritized, or polled conditions to indicate if an interrupt is pending.

The following figure illustrates the Interrupt Control functions, beginning with the highest priority and ending with the lowest priority.

Bits 5 4 3 2 1 0	Type	Cause	Interrupt Reset Control
0 0 0 1 1 0	Receiver Line Status	Overrun, Parity, or Framing Error or Break Interrupt	Read the Line Status Register
0 0 0 1 0 0	Received Data Available	Data in the Receiver Buffer or the Trigger Level Has Been Reached.	Read the Receiver Buffer Register or FIFO Register Drops Below the Trigger Level.
0 0 1 1 0 0*	Character Time-Out Indication	No Characters Have Been Removed From or Put Into the Receiver FIFO Register During the Last Four Character Times, and at Least 1 Character is in it at This Time.	Read the Receiver Buffer Register
0 0 0 0 1 0	Transmitter Holding Register Empty	Transmitter Holding Register is Empty	Read the Interrupt Identification Register or Write to Transmitter Holding Register
0 0 0 0 0 0	Modem Status	Change in Signal Status From Modem	Read the Modem Status Register

* FIFO Mode Only

Figure 21. Interrupt Control Functions

Line Control Register (Base c + 3)

The format of asynchronous communications is programmed through the Line Control register.

Bit	Description
7	Divisor Latch Access Bit
6	Set Break
5	Stick Parity
4	Even Parity Select
3	Parity Enable
2	Number of Stop Bits
1	Word Length Select, Bit 1
0	Word Length Select, Bit 0

Figure 22. Line Control Register (Base c + 3)

Bit 7 This bit is set to 1 to gain access to the divisor latches of the baud-rate generator. It is set to 0 to gain access to the Receiver Buffer, Transmitter Holding, or Interrupt Enable registers.

Bit 6 When this bit is set to 1, set break is enabled. The serial output is forced to the spacing state and remains there regardless of other transmitter activity. When this bit is set to 0, set break is disabled.

Bit 5 When bits 5, 4, and 3 are set to 1, the parity bit is sent and checked as a logical 0. When bits 5 and 3 are set to 1, and bit 4 is set to 0, the parity bit is sent and checked as a logical 1. If bit 5 is set to 0, stick parity is disabled.

Bit 4 When this bit and bit 3 are set to 1, an even number of logical 1s are transmitted and checked in the data word bits and parity bit. When this bit is set to 0, and bit 3 is set to 1, an odd number of logical 1s are transmitted and checked in the data word bits and parity bit.

Bit 3 When set to 1, a parity bit is generated (transmit data) or checked (receive data) between the last data-word bit and stop bit of the serial data. (The parity bit produces an even or odd number of 1s when the data-word bits and the parity bit are summed).

Bit 2 This bit, with bits 1 and 0, specifies the number of stop bits in each serial character sent or received, as shown in the following figure.

Bits 2 1 0	Number of Stop Bits	Word Length
0 0 0	1	5 Bits
0 0 1	1	6 Bits
0 1 0	1	7 Bits
0 1 1	1	8 Bits
1 0 0	1.5	5 Bits
1 0 1	2	6 Bits
1 1 0	2	7 Bits
1 1 1	2	8 Bits

The word length is specified by bits 1 and 0 in this register.

Figure 23. Stop Bits and Word Length

Bits 1, 0 These bits specify the number of bits in each serial character that is sent or received.

Modem Control Register (Base c + 4)

This 8-bit register controls the data exchange with the modem, data set, or peripheral device emulating a modem.

Bit	Description
7−5	Reserved = 0
4	Loop Mode
3	Out 2 (IRQ Output Control)
2	Out 1
1	Request-to-Send
0	Data-Terminal-Ready

Figure 24. Modem Control Register (Base c + 4)

Bits 7 − 5 These bits are reserved and always set to 0.

Bit 4 This bit provides a loopback feature for diagnostic testing of the serial port. When bit 4 is set to 1:

- Transmitter-serial-output is set to the marking state.
- Receiver-serial-input is disconnected.
- Output of the Transmitter Shift register is “looped back” to the Receiver Shift register input.

  Note: The Transmitter and Receiver Shift registers are not accessible NS16550 registers.
- The modem control inputs (CTS, DSR, DCD, AND RI) are disconnected.
- The modem control outputs (DTR, RTS, OUT 1, AND OUT 2) are internally connected to the four modem control inputs.
- The modem control output pins are forced inactive.

When the serial port is in the diagnostic mode, transmitted data is immediately received. This feature allows the system microprocessor to verify the transmit-data and receive-data paths of the serial port.

When the serial port is in the diagnostic mode, the receiver and transmitter interrupts are fully operational. The modem control interrupts are also operational, but their sources are the lower four bits of the Modem Control register instead of the four modem control input signals. The interrupts are still controlled by the Interrupt Enable register.

Bit 3 When this bit is set to 0, the IRQ signal is disabled, the IRQ signal is always disabled.

Bit 2 This bit is not used in a normal mode. In loop mode, its status is reported to bit 6 (RI) of the Modem Status register.

Bit 1 This bit controls the ‘-request to send’ signal (-RTS) modem control output. When this bit is set to 1, -RTS is active. When this bit is set to 0, -RTS is inactive.

Bit 0 This bit controls the ‘-data terminal ready’ signal (-DTR) modem control output. When this bit is set to 1, -DTR is active. When this bit is set to 0, -DTR is inactive.

Line Status Register (Base c + 5)

This 8-bit read-only register provides the system microprocessor with status information about the data transfer.

Note: Writing to this register can produce unpredictable results.

Bit	Description
7	Error in Receiver FIFO Register
6	Transmitter Shift Register Empty
5	Transmitter Holding Register Empty
4	Break Interrupt
3	Framing Error
2	Parity Error
1	Overrun Error
0	Data Ready

Figure 25. Line Status Register (Base c+5)

Bit 7 In FIFO mode, this bit indicates that a parity error, framing error, or break occurred. This bit is cleared when the Line Status register is read in the FIFO mode. It is set to 0 in the Character mode.

Bit 6 This bit is set to 1 to indicate the Transmitter Holding register and the Transmitter Shift register are both empty. This bit is set to 0 when either register contains a data character.

In the FIFO or DMA mode, this bit is set to 1 when the Transmitter FIFO register and the Transmitter Shift register are both empty.

Bit 5 This bit indicates that the controller is ready to accept the next character for transmission. This bit is set to 1 to indicate that a character was transferred from the Transmitter Holding register to the Transmitter Shift register. This bit is set to 0 when a character is written to the Transmitter Holding register.

This bit also causes the controller to issue an interrupt if the interrupt is enabled.

In the FIFO or DMA register, this bit is set to 1 when the Transmitter FIFO register is empty. It is set to 0 when at least one byte is written to the Transmitter FIFO register.

Bit 4 This bit is set to 1 to indicate the received data input is held in the spacing state for longer than a full-word transmission time (the total time of start bit + data bits + parity + stop bits). This bit is reset to 0 when the Line Status register is read.

When a break interrupt occurs, only one zero character is loaded into the Receiver FIFO register. The next character is loaded after the receiver serial input changes to the marking state and receives the next valid start bit.

Note: Bits 1 through 4 are the error conditions that produce a receiver-line-status interrupt whenever any of the corresponding conditions are detected and the interrupt is enabled.

Bit 3 This bit is set to 1 when the stop bit, following the last data bit or parity bit, is at a spacing level. This indicates that the received character did not have a valid stop bit (framing error). This bit is reset to 0 when the Line Status register is read.

Note: In the FIFO or DMA mode, the framing error (or parity error for bit 2) is associated with the particular character in the Receiver FIFO register that it applies to. The error is indicated to the system microprocessor when its associated character is at the top of the Receiver FIFO register.

Bit 2 This bit is set to 1 to indicate a parity error (the received character does not have the correct even or odd parity, as selected by the even-parity-select bit). This bit is reset to 0 when the Line Status register is read.

Bit 1 When set to 1, this bit indicates that data in the Receiver Buffer register was not read before the next character was transferred into the Receiver Buffer register, destroying the previous character. This bit is reset to 0 when the Line Status register is read.

If the FIFO or DMA mode data continues to fill the Receiver FIFO register beyond the trigger level, an overrun error occurs. The overrun occurs only after the Receiver FIFO register is full and the next character is completely received in the Receiver Shift register. An overrun error is indicated to the system microprocessor when it happens. The character in the Receiver Shift register is overwritten, but it is not transferred to the Receiver FIFO register.

Bit 0 This bit is the receiver data-ready indicator. It is set to 1 when a complete incoming character has been received and transferred into the Receiver Buffer register or the Receiver FIFO register. This bit is reset to 0 by reading the Receiver Buffer register or by reading all of the data in the Receiver FIFO register.

Modem Status Register (Base c+6)

This 8-bit register is used to monitor the current state of the control lines from the modem (or external device). Also, bits 3 through 0 indicate change information.

Bit	Description
7	Data-Carrier-Detect
6	Ring Indicator
5	Data-Set-Ready
4	Clear-to-Send
3	Delta-Data-Carrier-Detect
2	Trailing Edge Ring Indicator
1	Delta-Data-Set-Ready
0	Delta-Clear-to-Send

Figure 26. Modem Status Register (Base c+6)

Bit 7 This bit is the inverted '-data carrier detect' signal (-DCD) modem control input. If bit 4 of the Modem Control register is set to 1, this bit is equivalent to bit 3 in the Modem Control register.

Bit 6 This bit is the inverted '-ring indicator' signal (-RI) modem control input. If bit 4 of the Modem Control register is set to 1, this bit is equivalent to bit 2 in the Modem Control register.

Bit 5 This bit is the inverted '-data set ready' signal (-DSR) modem control input. If bit 4 of the Modem Control register is set to 1, this bit is equivalent to bit 0 in the Modem Control register.

Bit 4 This bit is the inverted '-clear to send' signal (-CTS) modem control input. If bit 4 of the Modem Control register is set to 1, this bit is equivalent to bit 1 in the Modem Control register.

Bit 3 When set to 1, this bit indicates that the '-data carrier detect' signal (-DCD) modem control input has changed state since the last time it was read by the system microprocessor.

Note: Whenever bit 0, 1, 2, or 3 is set to 1, a modem status interrupt is generated.

Bit 2 When set to 1, this bit indicates that the '-ring indicator' signal (-RI) modem control input has changed from an active condition to an inactive condition.

Bit 1 When set to 1, this bit indicates that the '-data set ready' signal (-DSR) modem control input has changed state since the last time it was read by the system microprocessor.

Bit 0 When set to 1, this bit indicates that the '-clear to send' signal (-CTS) modem control input has changed state since the last time it was read by the system microprocessor.

Scratch Register (Base c+7)

This register can be used by the system microprocessor as a temporary buffer or work area.

Enhanced Registers

The registers in this section are only available with the Type 3 and Type 4 controllers. These registers are:

- Enhanced Command Register
- Enhanced Interrupt ID Register
- Enhanced Function Register 1
- Enhanced Function Register 2
- Enhanced Function Register 3
- Character Compare Data Register
- Receive Character Count Register.

Enhanced Command Register (Base e + 0)

This write-only register is used to issue the new commands: Stop Sending, Start Sending, and Reset Error/Break Indicator. A write to the register causes the command to be executed.

Bits	Description
7 − 2	Reserved
1, 0	Command Bits

Figure 27. Enhanced Command Register

Bits 7 − 2 These bits are reserved and always written as 0 to allow future expansion of the command bits.

Bits 1,0 These command bits (CB1-CB0) enable Stop Sending and Start Sending.

CB1	CB0	Command
0	0	Reserved
0	1	Start Sending
1	0	Stop Sending
1	1	Reset Error/Break Indicator

Figure 28. Command Decode

Start Sending—This command starts or continues transmitting in the DMA or the FIFO mode. Characters in the FIFO mode are transmitted first. This command is used in the FIFO mode to restart the transmitter if it has been stopped by a Stop Sending command or by a Stop on Match function. In DMA mode, the transmit DMA request (TX DMA REQ) will not be active until the Start Sending Command is issued.

Stop Sending—This command empties the Transmitter Shift register and stops transmitting, regardless of the FIFO mode. After the transmitter is stopped, a character is written to the Transmitter Holding register. The character is then loaded into the Shift register and transmitted. After the character is written to the transmitter-holding-register-empty, the interrupt can be enabled. When the interrupt occurs, it signals that the send-single-character operation is complete. If multiple characters are written to the Transmitter Holding register, with the transmitter-holding register-empty interrupt enabled, multiple interrupts occur. To ensure that all characters have been transmitted, wait for the appropriate number of transmitter-holding register-empty interrupts before issuing the next Start Sending command to resume transmitting the FIFO register. There is no interrupt associated with this command. The shift register operation is not affected by this command.

Reset Error/Break—This command resets the Error/Break bit in the received-data-status byte, which is optionally stored with received data. The Reset command affects the next status byte, which is stored in the FIFO register.

The logic for controlling the Error/Break bit is at the input of the Receive FIFO. If there is an error for the current character being received, the corresponding status byte will have the Error/Break bit set. When the command to reset the Error/Break indicator is issued, the indicator is 0 in the next received-data-status byte placed in the FIFO register, unless that byte also has an error.

Check each byte that has the Error/Break bit set and issue a Reset command every time an error is found. This procedure should minimize the overhead associated with error detection.

Reserved Register (Base e + 1)

This is a reserved register.

Enhanced Interrupt ID Register (Base e + 2)

The pending interrupt is determined by decoding bits 1 through 5. For interrupt reset purposes, reading this register is equivalent to reading the Interrupt ID register.

Bits	Description
7 – 6	Reserved
5 – 1	Interrupt ID
0	Interrupt Pending

Figure 29. Enhanced Interrupt ID Register

Bits 7 – 6 These bits are reserved and are always set to 0.

Bits 5 – 1 These bits are the encoded IDs of the pending interrupt.

Bit 0 This bit indicates if an interrupt is pending. When it is a 0, an interrupt is pending, and bits 5 through 1 identify the interrupt. When it is a 1, no interrupt is pending.

The new interrupts have higher priority than the existing Type 2 controller interrupts. After a new interrupt has been enabled, interrupt priority exists. The following figure illustrates the Enhanced Interrupt Control functions, beginning with the highest priority and ending with the lowest priority.

Bits 5 4 3 2 1 0	**Type**	**Source**	**Interrupt Reset Control**
1 0 0 0 0 0	Receive TC	TC on DMA Receive	Read the Enhanced Interrupt Register
1 0 0 0 1 0	Transmit TC	TC on DMA Transmit	Read the Enhanced Interrupt Register
1 1 0 0 0 0	CC0 Match	Match on CC0	Read the Enhanced Interrupt ID Register
1 1 0 0 1 0	CC1 Match	Match on CC1	Read the Enhanced Interrupt ID Register
1 1 0 1 0 0	CC2 Match	Match on CC2	Read the Enhanced Interrupt ID Register
1 0 0 1 0 0	RCCR = 0	Receive Character Count	Read the Enhanced Interrupt ID Register
1 0 0 1 1 0	Transmitter Empty	THR and TSR Empty	Read the Enhanced Interrupt ID Register
0 0 0 1 1 0	Receiver Line Status	Overrun, Parity, or Framing Error or Break Interrupt	Read the Line Status Register
0 0 0 1 0 0*	Received Data Available	Data in the Receiver Buffer or the Trigger Level Has Been Reached.	Read the Receiver Buffer Register or FIFO Register Drops Below the Trigger Level.
0 0 1 1 0 0**	Character Time-Out Indication	No Characters Have Been Removed From or Put Into the Receiver FIFO Register During the Last Four Character Times	Read the Receiver Buffer Register in FIFO Mode. In DMA Mode Read EIIR to Reset.
0 0 0 0 1 0	Transmitter Holding Register Empty	Transmitter Holding Register is Empty	Read the Interrupt Identification Register or Write to Transmitter Holding Register
0 0 0 0 0 0	Modem Status	Change in Signal Status from Modem	Read the Modem Status Register

Note: *No trigger level interrupt in DMA Mode. ** FIFO and DMA Mode

Figure 30. Enhanced Interrupt Control Functions

Enhanced Function Register 1 (Base e + 3)

This register is a read and write register that enables new interrupts and DMA modes. A logical 1 enables the function and a logical 0 disables the function.

Bits	Description
7	DMA Receive
6	DMA Transmit
5	Enable Receive Data Status
4	Terminal Count Receive
3	Terminal Count Transmit
2	Stop Transmitter Line Error
1	Transmitter Empty
0	Receive Character Count Register

Figure 31. Enhanced Function Register 1 (Base e + 3)

Bit 7 This bit enables the DMA receive mode. The FIFO mode must be enabled (bit 0 in the FIFO Control register) before the DMA receive mode is enabled. The FIFO trigger level can be programmed, as appropriate, by writing to bits 6 and 7 in the FIFO Control register.

Bit 6 This bit enables the DMA transmit mode. The FIFO mode must be enabled (bit 0 in the FIFO Control Register) before the DMA transmit mode is enabled. Initially, a Start Sending command must be issued to start actual transmission. TX REQ cannot be generated until a Start Sending Command is issued.

Bit 5 This bit enables alternate bytes of data followed by Received Data status, to be stored in the Receive FIFO register. When this bit is set to 1, the Receive FIFO register has a capacity of eight data bytes plus eight status bytes. If a status byte is at the bottom of the FIFO register, the Line Status register indicates a good byte regardless of the status of the associated data byte. The enhanced-received-data-status bit should be set as part of the async port initialization. Toggling this bit while receiving serial data produces undefined results.

Note: No data is received during the time the FIFO register is cleared and the received-data status bit is toggled. Resetting this bit disables the storing of received data status.

Bit 4 This bit enables an interrupt when the terminal count is reached on a DMA receive operation. This signals that the last character of the last DMA buffer has been filled and that the FIFO register can fill and overrun if new DMA buffers are not allocated.

Bit 3 This bit enables an interrupt when the terminal count is reached on a DMA transmit operation. This signals that the last character in the last DMA buffer has been read into the FIFO register.

Bit 2 When set, this bit stops the transmitter after the Shift register empties on any received line error (OE, PE, FE, BI). The received line error is detected before the character is placed in the FIFO or DMA mode. The receiver continues to function normally. The transmitter can be restarted with the Start Sending command. If an interrupt is desired, the enable-line-status interrupt bit must be set in the Interrupt Enable register.

Bit 1 This bit enables an interrupt when the Transmit Hold register and Transmit Send register are empty in Character mode, or when the FIFO register and Transmit Send register are empty in the FIFO or DMA mode. The transmitter empty bit in the Line Status register is changed from 0 to 1.

Bit 0 This bit enables an interrupt when the Receive Character Count register is decremented to zero.

Enhanced Function Register 2 (Base e + 4)

This read and write register enables the transmitter controls, the modem pacing, and the baud-rate functions. A logical 1 enables the function and a logical 0 disables the function.

Bits	Description
7	Byte Pacing
6	Set High Frequency Rate
5	Set Slow Transmit Rate
4	Set Slow Receiver Rate
3	Receive the Receiver via DSR
2	Control the Transmitter via DCD
1	Control the Transmitter via DSR
0	Control the Transmitter via CTS

Figure 32. Enhanced Function Register 2 (Base e + 4)

| **Bit 7** This bit affects byte pacing. The byte pacing function allows the controller to support operations with a slow microprocessor. When this bit is set to 1, every byte of data is transmitted at a pacing rate of 256 times the receiver clocks value in the Receive Character Count register. An interrupt is generated when the Receive Character Count register reaches 0 and the Enhanced Function Register 1 equals 1. When this bit is set to 0, the Receive Character Count register is used for counting receiving characters if the Receive Character Count register is loaded with a value greater than 0.

Bit 6 This bit sets the high-frequency rate. When set to 1, this bit selects a a 11.0592 frequency.

Note: Applications that select the 11.0592 MHz rate should reset this bit to 0 when exiting.

This allows higher bit rates to be selected, up to a maximum rate of 345,600 bits per second on transmit and receive. When this bit is 1, the divisor latches must be set to 2 or greater.

Bit 5 This bit sets the slow transmit rate to 1/16 of the rate programmed in the baud-rate generator.

Bit 4 This bit sets the slow receiver rate. If this bit is set, the receiver rate is set to 1/16 of the rate programmed in the baud-rate generator.

Note: Bits 4, 5, and 6 should be set as part of the async port initialization. Toggling these bits during transmit and receive can produce undefined results.

Bit 3 This bit resets the receiver by issuing the '-data set ready' signal (-DSR, bit 5 of the Modem Status register). If this function is enabled, the receiver is turned off when -DSR equals 0 (bit 5 of the Modem Status register equals 0) and is turned on when -DSR equals 1 (bit 5 of the Modem Status register equals 1). If -DSR becomes inactive, the character currently being received is discarded.

Bit 2 This bit controls the transmitter by issuing the '-data carrier detect' signal (-DCD, bit 3 of the Modem Status register). If this function is enabled, the transmitter is turned off when the -DCD equals 0 (bit 3 of the Modem Status register), and is turned on if -DCD equals 1. However, the transmitter must be initially turned on by a Start Sending command (After a Start Sending command has been issued, the transmitter is turned on and off based on the state of the '-data carrier detect' signal).

Note: If the transmitter is stopped, the Transmitter Shift register is emptied but no additional characters are loaded from the Transmitter FIFO or Transmitter Hold register.

Bit 1 This bit controls the transmitter via the '-data set ready' signal (-DSR). If this function is enabled, the transmitter is turned off when -DSR equals 0, and is turned on when -DSR equals 1. However, the transmitter must be initially turned on by a Start Sending command (After a Start Sending command has been issued the transmitter is turned on and off based on the state of the '-data set ready' signal).

Bit 0 This bit controls the transmitter via the '-clear-to-send' signal. (-CTS, bit 4 of the Modem Status register). If this function is enabled, the transmitter is turned off when -CTS equals 0, and is turned on when -CTS equals 1. However, the transmitter must be initially turned on by a Start Sending command (After a Start Sending command has been issued the transmitter is turned on and off based on the state of the 'clear-to-send' signal).

Enhanced Function Register 3 (Base e + 5)

This read and write register is used to control the Character Compare registers. There are three 8-bit Character Compare registers and three 4-bit Character Compare Function registers. The Character Compare registers contain match characters and the Character Compare Function registers contain match functions. A Character Compare register and its Character Compare Function register can be programmed independently of the other Character Compare registers and character compare function registers.

To read from or write to the Character Compare registers, the address for the register must be written to Enhanced Function Register 3. Then the registers can be read from or written to, using the Character Compare register.

Bits	Description
7−3	Reserved
2−1	Character Compare Address Lines
0	Select Character Compare Register

Figure 33. Enhanced Function Register 3 (Base e+3)

Bits 7 – 3 These bits are reserved and are always set to 0.

Bits 2 – 1 These bits select the character compare address lines, the address of the character compare register, or the address of the Character Compare Function register.

Bit 0 This bit enables the Select Character Compare register. A logical 1 specifies the address in bits 1 and 2 for a Character Compare register. A logical 0 specifies the address for a Character Compare Function register.

The following table shows access to the Character Compare registers and the Character Compare Function registers for bits 2, 1 and 0.

Bits 2 1 0	**Function of Character Compare Data Register**	**Register Accessed**
0 0 0	Match Functions	Character Compare Function Register 0
0 1 0	Match Functions	Character Compare Function Register 1
1 0 0	Match Functions	Character Compare Function Register 2
0 0 1	Match Character	Character Compare Register 0
0 1 1	Match Character	Character Compare Register 1
1 0 1	Match Character	Character Compare Register 2

Figure 34. Access to CCRs and CCFRs

Character Compare Data Register (Base e + 6)

This register is used to read and write the match character for a Character Compare register, or the match function for a Character Compare Function register. The register is specified by first writing to Enhanced Function Register 3.

| The controller can be programmed to perform a specific operation | when a character match occurs. It can start the transmitter, stop the transmitter, delete the matched character from the incoming data stream, or generate an interrupt. Multiple match functions per Character Compare Function register are supported. If START and STOP are both set, then STOP takes precedence because they are mutually exclusive. Each character compare register is compared to the received data character, and if there is a match, then the programmed action takes place.

| If the controller is programmed to interrupt on a character match, then the interrupt occurs as soon as the match is detected. To disable a character compare register, clear the corresponding Character Compare Function register. The format of the Character

Compare Data register for the Character Compare Function register is shown on the following page.

Bits	Description
3	Start Transmitter on Character Match
2	Stop Transmitter on Character Match
1	Delete Character on Character Match
0	Interrupt on Character Match

Figure 35. Character Compare Register (Base e + 6)

Bit 3 A1 starts the transmitter when a match occurs. The transmitter does not start unless a Start Transmitter command has been issued previously.

Bit 2 A1 stops the transmitter when a match occurs.

Bit 1 A1 causes the character to be deleted when a match occurs.

Bit 0 A1 enables an interrupt when a match occurs. This interrupt occurs as soon as there is a match with a received data character.

The format of the Character Compare Data register for a character compare register is an 8-bit match character. If the word length is less than eight bits, the match character should be right justified and any unused bits should be set to 0 when written to the Character Compare Data Register.

Receive Character Count Register (Base e + 7)

This 8-bit register is used in byte pacing and receive character count functions. The Receive Character Compare register is decremented when a character is read from the Receive FIFO register during a receive character count operation, or at every 256 receiver clocks during Byte Pacing operation.

If the interrupt on Receiver Character Count is set (Enhanced Function Register 1, Bit 0), then an interrupt is generated when the Receive Character Count register is decremented to 0, regardless of any operation.

Because this register is a countdown counter and does not wrap around when reaching zero, the user must load a value to the Receive Character Count register before using it. When read, this register contains the current count (the count cannot be exact since the receiver or 'receiver clocks' cannot be stopped to read this register).

Signal Descriptions

Modem-Control Input Signals

The following are input signals from the modem or external device to the controller. Bits 7 through 4 in the Modem Status register indicate the condition of these signals. Bits 3 through 0 monitor these signals to indicate when the modem changes state.

-Clear to Send (-CTS)

When active, this signal indicates that the modem is ready for the serial port to transmit data.

-Data Set Ready (-DSR)

When active, this signal indicates that the modem or data set is ready to establish the communications link and transfer data with the controller.

-Ring Indicator (-RI)

When active, this signal indicates that the modem or data set detected a telephone ringing signal.

-Data Carrier Detect (-DCD)

When active, this signal indicates that the modem or data set detected a data carrier.

Modem-Control Output Signals

The following are controller output signals. All are set inactive by a master reset operation. These signals are controlled by bits 3 through 0 in the Modem Control register.

-Data Terminal Ready (-DTR)

When active, this signal informs the modem or data set that the controller is ready to communicate.

-Request to Send (-RTS)

When active, this signal informs the modem or data set that the controller is ready to send data.

Voltage Interchange Information

The signal is considered in the *marking* condition when the voltage on the interchange circuit, measured at the interface point, is more negative than -3 V dc, with respect to signal ground. The signal is considered in the *spacing* condition when the voltage is more positive than +3 V dc with respect to signal ground. The region between +3 V dc and -3 V dc is defined as the transition region and is considered an invalid level. Voltages more negative than -15 V dc or more positive than +15 V dc are also considered as invalid levels.

Interchange Voltage	Binary State	Signal Condition	Interface Control Function
Positive Voltage	Binary 0	Spacing	On
Negative Voltage	Binary 1	Marking	Off

Figure 36. RS-232 Voltage Levels

Extended Performance Requirements

Extended performance applies to Type 3 and Type 4 controllers only. Although the serial port is compatible with EIA-232-D at speeds up to 20,000 bits per second, there are additional requirements for operating the port at speeds up to 345,600 bits per second. These requirements fall into two areas, the interconnection cable and the attached equipment.

The cable should not be longer than 20 feet when operating at extended performance speeds. Each signal wire has an individual shield and should have capacitance between 100 and 450 picofarads to the shield. All individual shields are connected to pin 7 at each end. These requirements can be met by using one IBM Personal Computer Communications Adapter Cable for 10 feet, or two connected in series for 20 feet.

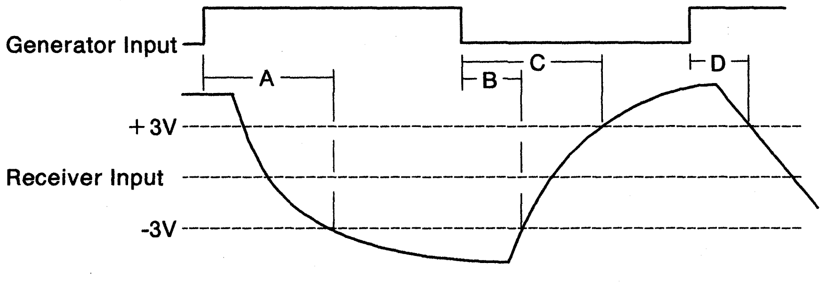
The attached equipment should meet the following requirements:

- The capacitance for an input or output signal should be less than or equal to 120 picofarads.
- The timing oscillator accuracy is ±.01%.
- The maximum generator skew equals 970 nanoseconds with the cable attached and a capacitive load (including cable) between 100 and 690 picofarads.
- The maximum receiver skew equals 160 nanoseconds.
- The receiver deserializer should decode the data stream by sampling a minimum of 16 times per unit interval such as is done in the NS8250 and similar Universal Asynchronous Receive Transmitters (UARTs).

To specify the maximum distortion in the signals, the concept of skew is introduced. The receiver skew is defined as the worst case difference in delay between the rising and falling edges from a 12V peak-to-peak (P-P) square wave connected to the input of the receiver to when it reaches the deserializer. For example, if the rising edge delay is 200 nanoseconds and the falling edge delay is 120 nanoseconds then the skew is 200 nanoseconds minus 120 nanoseconds, which equals 80 nanoseconds. The 80 nanoseconds is well within the 160-nanosecond requirement.

The generator skew is measured using the sending serializer to generate a square wave and measuring the delays at the input of an attached receiver with the cable and all capacitive loadings (690 picofarad maximum) included.

The generator waveforms graphic shown below illustrates where the measurements are taken. The generator skew is A-B or C-D, whichever is larger.



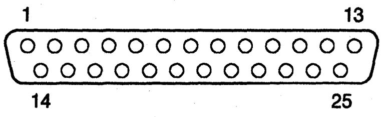
Generator Waveforms

Connectors

The serial port provides a standard 25-pin male D-shell connector with pin assignments defined for RS-232C and may provide a 9-pin male D-shell connector.

Current loop is not supported.

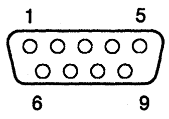
The following is the 25-pin signal and pin assignment for the serial port in a communications environment.



Pin	Signal	Pin	Signal
1	Not Connected	14	Not Connected
2	Transmit Data	15	Not Connected
3	Receive Data	16	Not Connected
4	Request to Send	17	Not Connected
5	Clear to Send	18	Not Connected
6	Data Set Ready	19	Not Connected
7	Signal Ground	20	Data Terminal Ready
8	Data Carrier Detect	21	Not Connected
9	Not Connected	22	Ring Indicator
10	Not Connected	23	Not Connected
11	Not Connected	24	Not Connected
12	Not Connected	25	Not Connected
13	Not Connected		

Figure 37. Serial Port Connector Signal and Pin Assignments (25-Pin)

The following is the 9-pin signal and pin assignments. This connector supports the RS-232 interface.



Pin	Signal
1	Data Carrier Detect
2	Receive Data
3	Transmit Data
4	Data Terminal Read
5	Signal Ground
6	Data Set Ready
7	Request To Send
8	Clear To Send
9	Ring Indicator

Figure 38. Serial Port Connector Signal and Pin Assignments (9-Pin)

Index

Parallel Port Controller

Figures

Description

The parallel port allows the attachment of devices that transfer eight bits of parallel data at standard transistor-transistor levels. It has a 25-pin, D-shell connector. The primary function of the parallel port is to attach a printer with a parallel interface to the system. The parallel port function consists of the controller and the connectors.

There are three types of parallel port controllers: Type 1, Type 2, and Type 3. The type number indicates the function provided in the controller. The address of each controller can be configured and set to 1, 2, or 3 for Type 1 and Type 2 controllers. The address can be set to 1, 2, 3, or 4 for Type 3 controller.

The parallel port controller has an extended mode that supports bidirectional input and output. All three types support bidirectional input and output. In addition, Type 2 and Type 3 parallel port controllers support DMA transfers when in this mode. During power-on self-test the Type 2 and Type 3 controllers are put into extended mode so that DMA is available (See Programming Considerations on page 3). The parallel port controller supports level-sensitive interrupts and a readable interrupt-pending status indicator.

The following figure is a block diagram of the parallel port controller.

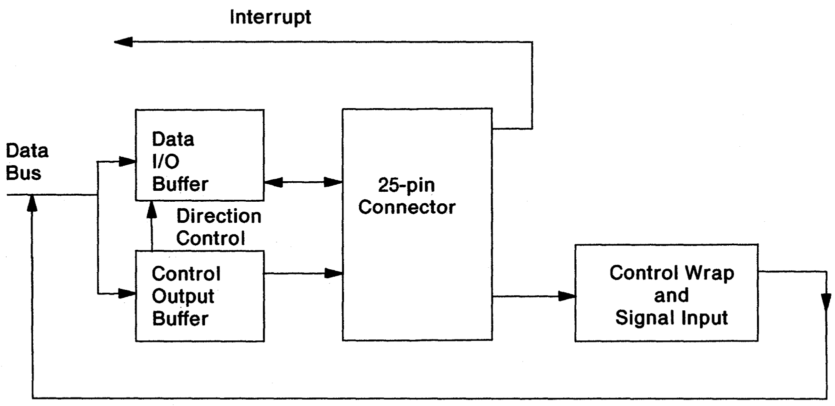


Figure 1. Parallel Port Controller

Programmable Option Select

This section describes the address selection, the extended mode, and DMA arbitration for the parallel port controllers.

Address Selection

The parallel port can be configured to the customary three addresses used by IBM Personal Computer products. These addresses are selected through programmable option select (POS) registers during system board setup.

The address assignments for each configuration are shown in the following tables. All addresses are in hexadecimal.

Port Number	Parallel Data Register	Device Status Register	Device Control Register	Reserved Register	IRQ
Parallel 1	03BC	03BD	03BE	03BF	7
Parallel 2	0378	0379	037A	037B	7
Parallel 3	0278	0279	027A	027B	7

Figure 2. Address Assignments (Type 1)

Port Number	Parallel Data Register	Device Status Register	Device Control Register	Interface Control Register	Interface Status Register	Reserved Register	IRQ
Parallel 1*	03BC	03BD	03BE	03BF			7
Parallel 2	0378	0379	037A	037B	037C	037D	7
Parallel 3	0278	0279	027A	027B	027C	027D	7

*Type 2 controllers do not support DMA on parallel port 1.

Figure 3. Address Assignments (Type 2)

Port Number	Parallel Data Register	Device Status Register	Device Control Register	Interface Control Register	Interface Status Register	Reserved Register	IRQ
Parallel 1*	03BC*	03BD*	03BE*	03BF*			7
Parallel 1	1278	1279	127A	127B	127C	127D	7
Parallel 2	0378	0379	037A	037B	037C	037D	7
Parallel 3	0278	0279	027A	027B	027C	027D	7
Parallel 4	1378	1379	137A	137B	137C	137D	7

*These addresses are for compatibility. To use full function, set the address range to start at hex 1278.

Figure 4. Address Assignments (Type 3)

Extended Mode

The extended mode option is selected through the POS function during system board setup. The extended mode makes the parallel port an 8-bit parallel bidirectional interface and makes DMA available for Type 2 and Type 3 controllers. The parallel port provides half duplex transfers when in bidirectional operation mode. Direction is determined by bit 5 of the Device Control register and DMA is controlled through the Interface Control register.

Arbitration Level (Type 2 and Type 3)

The Type 2 controller is fixed at arbitration level 6 and cannot be set through POS. The Type 3 controller allows arbitration levels to be configured to any level through the POS function.

Parallel Port Controller Programming Considerations

The following are some considerations for programming the parallel port controller.

The interface has three registers that respond to input and output instructions. The three registers are:

- A read and write Parallel Data register
- A read only Device Status register
- A read and write Device Control register.

The Type 2 and Type 3 controllers have three additional registers:

- A read and write Interface Control register
- A read only Interface Status register
- A write only Reserved register.

These registers are available in extended mode only.

During POST the parallel port is configured as an output port. POST status information is written to this port during the power-on initialization or the initialization caused by a reset from the keyboard (Ctrl + Alt + Del).

DMA Mode (Type 2 and Type 3 Only)

Single DMA transfers are supported both when sending and receiving. The DMA enable bit in the Interface Control register is set to 1 when DMA service is requested.

The parallel port has two idle states while it is in the DMA mode:

- Not ready (end-of-data bit = 1): Any request generates an interrupt if the interrupt is enabled. This state is entered after completing a DMA transfer or after setting the End of Data bit (bit 6) in the Interface Control register to 1.
- Ready (end-of-data bit = 0): While in this state, a pulse on the '-acknowledge' signal starts a DMA transfer. This state is entered after the Start DMA bit or the Reset End of Data bit (bit 6) is set in the Interface Control register to 1.

In addition, there are two data transfer states:

- Send byte: This state is entered when bit 5 of the Device Control register is set to 0 and a Start DMA or an '-acknowledge' signal is set. A DMA fetch transfer is initiated, the data is placed on the interface, and the 'strobe' signal line is pulsed.
- Receive byte: This state is entered when the direction bit is set to 1 and an '-acknowledge' signal is set. A DMA store is initiated and the 'strobe' signal is pulsed to acknowledge the completion of the operation. The incoming data is latched at the trailing edge of the '-acknowledge' signal.

Sending

Writing a 1 to the start-DMA bit in the Interface Control register initiates the DMA transfer to the attached device. (The enable DMA bit must have been set to 1 in a previous operation). The DMA Controller channel and the direction of the transfer are set before writing a 1 to the start-DMA bit. If the device is not busy, the parallel port controller requests the system bus. After control of the bus is gained, the following sequence of events occur:

1. Data is read from memory and written to the Parallel Data register by the DMA controller.
2. Data is carried to the attached device using the 'strobe' signal on the parallel interface.
3. The 'strobe' signal is activated automatically after the Parallel Data register is written.
4. The device issues the '-acknowledge' signal when the data transmission is completed.

Note: The '-acknowledge ' signal is used to trigger the next DMA transfer, if the end-of-data latch (bit 6 of the Interface Status register equals 0) is reset and DMA is enabled. In addition, the attached device must hold the 'busy' signal inactive for the parallel port to issue a DMA request.

When a terminal count is reached during a DMA transfer, the end-of-data latch is set and the next DMA transfer is prevented. It is known that the DMA controller has transferred the last byte of data to the parallel port controller. However, it is not known if the attached device has received the last byte of data. The parallel port controller is designed to interrupt at the '-acknowledge' signal after the end-of-data latch is set. At the completion of a DMA transfer, the parallel port controller will interrupt if the terminal-count interrupt-enable bit is set.

By setting the end-of-data (EOD) bit in the Interface Control register, the current DMA operation will be terminated. It should be noted that no interrupt will be generated by setting this bit.

This causes an effect similar to terminal count. One additional byte of data may be sent depending on the timing of the setting of the bit.

Disabling the DMA while a DMA transfer is in progress stops the generation of 'strobe' signals and causes an interrupt to end the DMA transfer.

Receiving

The DMA channel must be initialized prior to receiving data from the attached device. When the parallel port controller is ready to receive data from the attached device, the parallel port controller uses the '-acknowledge' signal. The '-acknowledge' signal initiates the DMA transfer to the system memory, if the end-of-data latch is reset and DMA is enabled. Data transmission allows the parallel port controller to request the system bus. After control of the bus is gained, the following sequence of events occur:

1. Data in the Parallel Data register of the parallel port controller is read and then written to memory by the DMA controller.
2. The 'strobe' signal is activated after the read command pulse to the Parallel Data register.

The parallel port controller cannot receive subsequent data until the previous data has been read by the DMA controller and transferred to the system memory. When the '-acknowledge' signal goes low, (data is being placed on the bus) the parallel port controller asserts the '-autofeed' signal (-AUTOFD) line high to inform the attached device of a busy state. After the DMA controller reads the data from the Parallel Data register, a 'strobe' signal is generated automatically as an acknowledgement of data transmission. A positive edge of the 'strobe' signal resets the -AUTOFD line low to tell the attached device that the controller is not busy. The 'busy' signal has no effect in receive mode.

To determine the end of a DMA transfer, the parallel port controller interrupts if the terminal-count interrupt-enable bit is on. When the terminal count is reached during a DMA transfer, the end-of-data latch is set to prevent the next DMA cycle and an interrupt occurs. The DMA controller has transferred the last byte of data to the memory.

Interrupt Condition

When enabled, the following event can cause an interrupt:

- Any transition of the '-error' signal
- Any transition of the 'paper end' signal
- Any transition of the 'select' signal

- The positive edge of the '-acknowledge' signal
  - When DMA is disabled
  - When DMA is enabled and end-of-data is set
- Terminal count while DMA is enabled while receiving.

Output Data Rate

The approximate time between two data bytes in a DMA operation is 5 microseconds, assuming that the attached device returns the '-acknowledge' signal to the 'strobe' signal immediately. In addition, the '-acknowledge' signal is 1.0 microseconds wide (the '-acknowledge' signal falls low and not later than approximately 1.0 microseconds after the rising edge of the '-strobe' line). To calculate the data rate, the delay from the '-strobe' signal to the '-acknowledge' signal on the attached device side should be added for each transmission. The transfer rate varies according to the DMA channel usage of the system.

Register Definitions

The following definitions apply to all types of parallel port controllers, unless specified otherwise. All reserved bits will be written as 0, and read as a 1.

Parallel Data Register

The Parallel Data register is a read and write register. Its output drivers are enabled by the direction bit in the Device Control register. A read operation returns the output data, if the output drivers are enabled or if data is read from the attached device when the drivers are not enabled. Care must be taken not to enable the drivers when the attached device is also driving the interface.

CAUTION:
Damage to the system can occur if the controller and the attached device are driving data onto the data lines at the same time. Therefore, do not enable the drivers when the attached device is driving the interface.

The Parallel Data register is an 8-bit data register for both the compatible and extended modes. In compatible mode, writing to this register immediately presents data to the connector. Reading this register returns the last byte written.

In extended mode, writing to this register latches the data; however, the data is presented to the connector only if the direction bit is 0 (write). Reading the register returns either:

- The last byte written, if the direction bit is 0
- The data on the connector from the attached device, if the direction bit is 1.

Bit	Description
7 - 0	Data

Figure 5. Parallel Data Register

Bits 7 - 0 These bits represent the data (D7 - D0) on the signal lines of the connector.

Device Status Register

This read-only register contains the status of the attached device and the status of the interrupt.

Note: Reading this register resets an interrupt latch caused by the '-acknowledge' signal. Because reading this register restores bit 2 (-IRQ STATUS) to 1, there is a small window where the rising edge of the 'acknowledge' signal can arrive at the same time that the register is being read. This prevents bit 2 from being set to 0 as it would have been without the read operation. This condition also prevents the associated interrupt if it was enabled in the Device Control Register.

This register returns the interrupt-pending status of the interface, and the real-time status of the connector pins, as shown in figure 6.

Bit	Description
7	-BUSY
6	-ACKNOWLEDGE
5	PAPER END
4	SELECT
3	-ERROR
2	-IRQ STATUS
1, 0	Reserved

Figure 6. Device Status Register

Bit 7	This bit represents the state of the '-busy' signal. When this bit is set to 0, the printer is busy and cannot accept data.
Bit 6	This bit represents the current state of the device '-acknowledge' signal. When this bit is pulsed, the device has received a character and is ready to accept another.
Bit 5	This bit represents the current state of the device 'paper end' signal. When this bit is set to 1, the printer has detected the end of the paper.
Bit 4	This bit represents the current state of the 'select' signal. When this bit is set to 1, the printer has been selected.
Bit 3	This bit represents the current state of the device '-error' signal. When this bit is set to 0, the printer has encountered an error condition.
Bit 2	This bit is set to 0 when the device has acknowledged the previous transfer using the '-acknowledge' signal. This bit is set to 1 when the Device Status register or the Interface Status register is read. An interrupt is pending when this bit is set to 0. This bit is used in non-DMA mode only.
Bits 1, 0	These bits are reserved.

Device Control Register

This is a read and write register that controls lines to the attached device. Reading this register returns the last byte written to the register, if the line is not driven by the attached device. The direction bit can be updated at any time, but the effect of the bit is masked when in non-extended mode (in compatible mode the direction is always out, even if the bit reads as 1).

Bit	Description
7, 6	Reserved
5	Direction
4	IRQ EN
3	SLCT IN
2	-INIT
1	AUTO FD XT
0	STROBE

Figure 7. Device Control Register (Type 1 and Type 2)

Bits 7, 6 These bits are reserved and must be set to 0. This bit always reads as 1.

Bit 5 This bit controls the direction of the data port. When this bit is set to 0, the data drivers are enabled and the parallel data is placed on the output data lines. A read operation will return a 1 for Type 1 and the last value written for Type 2.

Bit 4 This bit enables the parallel port interrupt. When this bit is set to 1, an interrupt occurs when the '-acknowledge' signal changes from active to inactive (non-DMA mode only).

Bit 3 This bit controls the 'select in' signal. When this bit is set to 1, the printer is selected.

Bit 2 This bit controls the 'initialize' signal. When this bit is set to 0, the printer will be initialized.

Bit 1 This bit controls the 'automatic feed XT' signal. When this bit is set to 1, the printer automatically spaces the paper up one line for every carriage return.

Bit 0 This bit controls the 'strobe' signal to the printer. When this bit is set to 1, data is clocked into the printer.

Bit	Description
7	Autostrobe
6	Reserved
5	Direction
4	IRQ Enable
3	SLCT IN
2	-INIT
1	AUTO FD XT
0	STROBE

Figure 8. Device Control Register (Type 3)

Bit 7 This bit, when set to 1, enables the 'Autostrobe' signal.

Bit 6 This bit is reserved and must be set to 1. A read operation will return a value of 1.

Bit 5 This bit controls the direction of the data register. When this bit is set to 0, the data drivers are enabled and the parallel data is placed on the output data lines. A read operation will return a 1 in non-extended mode and the last value written in extended mode.

Bit 4	This bit enables the parallel port interrupt. When this bit is set to 1, an interrupt occurs when the '-acknowledge' signal changes from active to inactive (non-DMA mode only).
Bit 3	This bit controls the 'select in' signal. When this bit is set to 1, the printer is selected.
Bit 2	This bit controls the 'initialize' signal. When this bit is set to 0, the printer will be initialized.
Bit 1	This bit controls the 'automatic feed XT' signal. When this bit is set to 1, the printer automatically spaces the paper up one line for every carriage return.
Bit 0	This bit controls the 'strobe' signal to the printer. When this bit is set to 1, data is clocked to the printer.

CAUTION:
Do not enable the drivers for the data lines while the attached device is driving them.

Interface Control Register

This register controls the various functions available in the interface. All bits read as 1 in non-extended mode. Bits 1, 6, and 7 always read as 1.

Bit	Description
7	Start DMA
6	Reset EOD
5	Enable TC/ACK Interrupt
4	SLCT IRQ Enable
3	-ERROR IRQ Enable
2	PE IRQ Enable
1	Set EOD
0	Enable DMA

Figure 9. Interface Control Register (Type 2 and Type 3)

Bit 7	Writing a 1 to this bit initiates a DMA transfer. The Enable DMA (bit 0) must have been set to 1 in a previous write operation, otherwise the results are indeterminate. Reading this bit always returns as a 1.
Bit 6	Setting this bit to 1 resets the end-of-data latch, which allows the port to honor DMA transfer requests by the '-acknowledge' signal. Reading this bit always returns a 1.

Bit 5	Setting the bit to 1 enables interrupts to occur whenever the terminal count is achieved, or whenever an '-acknowledge' signal occurs with the end-of-data latch set. A 0 clears the terminal count '-acknowledge' signal in the Interface Status register and removes any pending interrupts caused by the terminal count. This bit is used only in DMA mode.
Bit 4	This bit is valid in the extended mode and setting this bit to 1 enables interrupts to occur on either edge of SELECT. Resetting this bit clears the 'select interrupt request' signal (SLCT IRQ) in the Interface Status register.
Bit 3	This bit is valid in the extended mode and enables interrupts to occur on either edge of an '-error' signal. Resetting this bit clears the 'error interrupt request' signal (-ERROR IRQ) in the Interface Status register.
Bit 2	This bit is valid in the extended mode and enables interrupts to occur on either edge of a 'paper end' signal. Resetting this bit clears the 'paper end interrupt request' (PE IRQ) in the Interface Status register.
Bit 1	Setting this bit to 1 sets the end-of-data latch which will stop DMA transfers before terminal count is reached. Reading this bit always returns a 1. This output generates a pulse and always reads as 1.
Bit 0	This bit enables the DMA function. Whenever this bit is set from a reset condition, the end-of-data latch (bit 1) must also be set to prevent unknown states. The extended mode must be set in the POS register before this bit can be set.

Writing certain bit combinations of the Interface Control register may cause unexpected results. Those bit combinations listed as reserved in the following figure, must not be used.

Bits 7 6 1 0	Function
00 00	Reserved
00 01	No Change to DMA Operation
00 10	Disable DMA
00 11	Halt DMA or Enable DMA
01 00	Reserved
01 01	Set Ready-to-Start DMA in Receive Mode
01 10	Reserved
01 11	Reserved
10 00	Reserved
10 01	Start DMA in Send Mode
10 10	Reserved
10 11	Reserved
11 00	Reserved
11 01	Reserved
11 10	Reserved
11 11	Reserved

Figure 10. Interface Control Register DMA Functions

Interface Status Register

This is a read-only register and is used to convey the status of the parallel interface. Reading this register resets the interrupt pending status bits and resets the interrupt request. The interrupt handler must save the status or process it completely before returning. Disabling the interrupts with the Interface Control register also clears the corresponding interrupt-request bit. All bits in this register read as 1 in compatible mode.

Note: Because reading the register resets the interrupt status (bits 2 through 5), there is a small window where the interrupting condition, if it occurs at the end of the read operation, will not set the corresponding interrupt status bit (the interrupt is also prevented). A periodic check of the corresponding status bits in the Device Status register and the EOD bit in this register can be used to detect this condition.

Bit	Description
7	Reserved
6	EOD
5	TC/ACK Interrupt
4	SLCT Interrupt
3	Error Interrupt
2	PE Interrupt
1,0	Reserved

Figure 11. Interface Status Register (Type 2 and Type 3)

Bit	
Bit 7	This bit is reserved.
Bit 6	When the EOD bit is 1, it indicates that the end-of-data latch is set and the parallel port is not ready to perform a DMA transfer. This occurs when the terminal count is reached or when the latch is set through the set-end-of-data-latch bit in the Interface Control register.
Bit 5	When the TC/ACK interrupt bit is set to 1, it indicates that the pending interrupt is caused by the terminal count or by an '-acknowledge' signal. If the EOD bit is 1, the interrupt was caused by the terminal count, otherwise it is caused by the 'acknowledge' signal.
Bit 4	When the SLCT interrupt bit is 1, it indicates that the pending interrupt is caused by any transition of the 'select' signal.
Bit 3	When the error interrupt bit is 1, it indicates that the pending interrupt is caused by any transition of the 'error' signal.
Bit 2	When the PE interrupt bit is set to 1, it indicates that the pending interrupt is caused by any transition of the 'paper end' signal.
Bit 1,0	These bits are reserved.

Reserved Register Initialization

This register must be loaded with a value of Hex 16 before using the DMA mode.

Reading this register gives unpredictable values.

Bit	Description
7 - 0	Must Be Hex 16

Figure 12. Reserved Register (Type 2 and 3)

Parallel Port Timing

Timing for the parallel port depends on the devices connected to the port. The following figure shows the minimum requirements for the parallel-port signal timing and any device that may be attached to the parallel port.

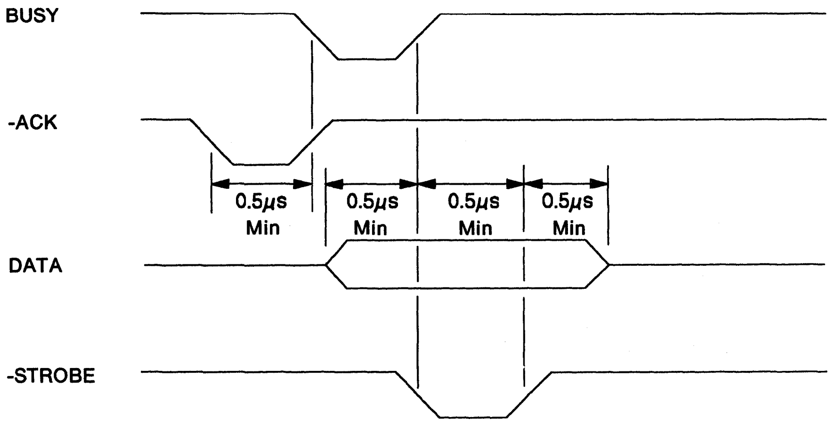


Figure 13. Parallel-Port Timing Sequence

The DMA (Types 2 and 3) and Autostrobe (Type 3) functions will generate these timings automatically. In sending modes, -STROBE will be activated 1.0 ±0.25 microseconds after the data is placed on the output data lines. The -STROBE pulse width is 1.0 ±0.25 microseconds. In DMA receive mode, a strobe pulse 1.0 ±0.25 microseconds wide will be generated as soon as the data has been read from the data lines.

For specific signal timing parameters, refer to the specifications for the equipment connected to the parallel port connector.

Signal Descriptions

The following figures, schematics, and tables, are representative circuits of the Parallel Port interface signals.

Parameter	Value	Limit
High-Level Input Voltage	2 V	Minimum
Low-Level Input Voltage	0.8 V	Maximum
High-Level Input Current	40 µA	Maximum
Low-Level Input Current	-0.8 mA	Maximum

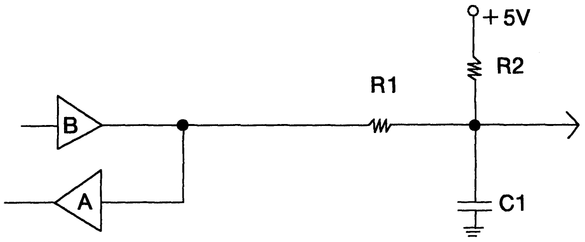
Figure 14. Receiver A Specifications

Parameter	Value	Limit
Sink Current	24 mA	Maximum
Source Current	-2.6 mA	Maximum
High-Level Output Voltage	2.4 Vdc	Minimum
Low-Level Output Voltage	0.5 Vdc	Maximum

Figure 15. Driver B Specifications

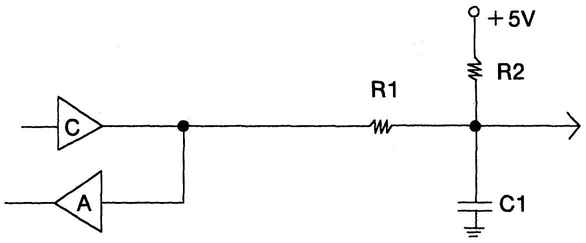
Parameter	Value	Limit
Sink Current	20 mA	Maximum
Source Current	Open Collector	
High-Level Output Voltage	Open Collector	
Low-Level Output Voltage	0.5 Vdc	Maximum

Figure 16. Driver C Specifications



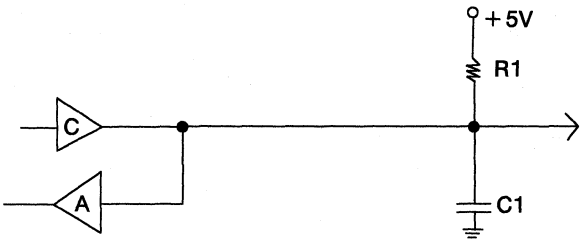
Symbol	Value
R 1	33 Ω
R 2	2 kilo ohms or Not Present
C 1	0.0022 μF or Not Present

Figure 17. Data Lines



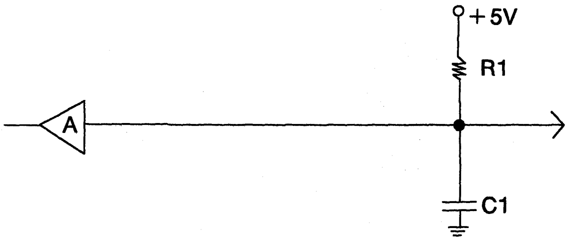
Symbol	Value
R 1	33 Ω
R 2	2 kilo ohms to 4.7 kilo ohms
C 1	0.0022 μF or Not Present

Figure 18. -STROBE



Symbol	Value
R 1	2 kilo ohms to 4.7 kilo ohms
C 1	0.0022 μF or Not Present

Figure 19. -AUTOFDXT, -INIT, -SLCTIN



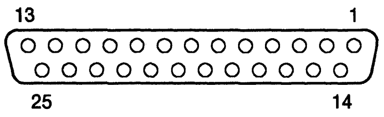
Symbol	Value
R 1	1 kilo ohms to 10 kilo ohms or Not Present
C 1	0.00068 to .0022 μF or Not Present

Figure 20. -ACK, BUSY, PE, SLCT, -ERROR

Connector

The parallel port connector is a standard 25-pin female D-shell connector.

The following figure shows the signal and pin assignments for the parallel port connector.



Pin No.	I/O	Signal Name	Pin No.	I/O	Signal Name
1	I/O	-STROBE	14	O	-AUTO FD XT
2	I/O	Data 0	15	I	-ERROR
3	I/O	Data 1	16	O	-INIT
4	I/O	Data 2	17	O	-SLCT IN
5	I/O	Data 3	18	NA	Ground
6	I/O	Data 4	19	NA	Ground
7	I/O	Data 5	20	NA	Ground
8	I/O	Data 6	21	NA	Ground
9	I/O	Data 7	22	NA	Ground
10	I	-ACK	23	NA	Ground
11	I	BUSY	24	NA	Ground
12	I	PE	25	NA	Ground
13	I	SLCT			

Figure 21. Parallel Port Connector Signal and Pin Assignments

Index

Notes:

Video Subsystem

Figures

Notes:

Section 1. Introduction

Video Subsystem

The system video can be generated by a Type 1 or Type 2 video subsystem:

- Type 1 video—Video Graphics Array (VGA)
- Type 2 video—Extended Graphics Array (XGA*).

Type 1

The Type 1 video contains the VGA function. The capabilities and operation of the VGA function are described in Section 2, "VGA Function" on page 2-1.

Only one Type 1 video subsystem is allowed in a system.

Type 2

The Type 2 video contains the XGA function, which supports the VGA mode, 132-column text mode, and extended graphics mode. The capabilities and operation of the XGA function are described in Section 3, "XGA Function" on page 3-1.

* XGA is a trademark of International Business Machines Corporation.

Section 2. VGA Function

Notes:

VGA Function Introduction

The basic system video is generated by the Type 1 or Type 2 video subsystem. The circuitry that provides the VGA function includes a video buffer, a video digital-to-analog converter (DAC), and test circuitry. Video memory is mapped as four planes of 64Kb by 8 bits (maps 0 through 3). The video DAC drives the analog output to the display connector. The test circuitry determines the type of display attached, color or monochrome.

The video subsystem controls the access to video memory from the system and the cathode-ray tube (CRT) controller. It also controls the system addresses assigned to video memory. Up to three starting addresses can be programmed for compatibility with previous video adapters.

In the graphics modes, the mode determines the way video information is formatted into memory, and the way memory is organized.

In alphanumeric modes, the system writes the ASCII character code and attribute data to video memory maps 0 and 1, respectively. Memory map 2 contains the character font loaded by BIOS during an alphanumeric mode set. The font is used by the character generator to create the character image on the display.

Three fonts are contained in read-only memory (ROM): an 8-by-8 font, an 8-by-14 font, and an 8-by-16 font. Up to eight 256-character fonts can be loaded into the video memory map 2; two of these fonts can be active at one time, allowing a 512-character font.

The video subsystem formats the information in video memory and sends the output to the video DAC. For color displays, the video DAC sends three analog color signals (red, green, and blue) to the display connector. For monochrome displays, BIOS translates the color information in the DAC, and the DAC drives the summed signal onto the green output.

The auxiliary video connector allows video data to be passed between the video subsystem and an adapter plugged into the channel connector.

When it is disabled, the video subsystem will not respond to video memory or I/O reads or writes; however, the video image continues to be displayed.

Note: Compatibility with other hardware is best achieved by using the BIOS interface or operating system interface whenever possible.

The following is a diagram of the VGA function.

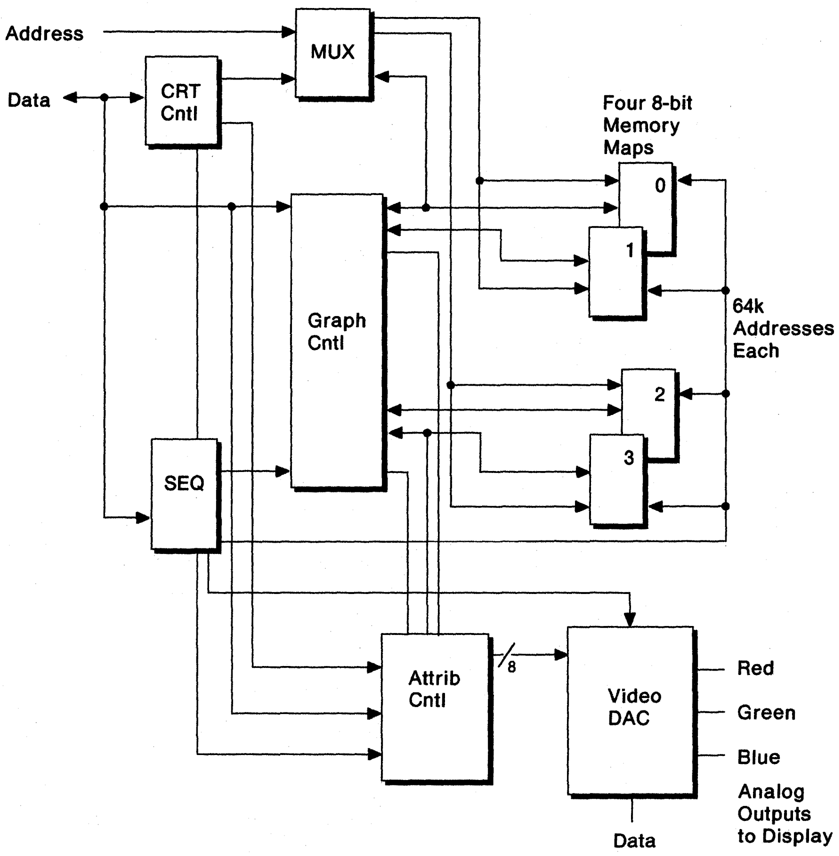


Figure 2-1. Diagram of the VGA Function

Major Components

The video subsystem contains all circuits necessary to generate the timing for the video memory and generates the video information going to the video DAC. The major components are: ROM BIOS, the support logic, and the Video Graphics Array interface.

ROM BIOS

BIOS provides software support and contains the character fonts and the system interface to run the video subsystem.

Support Logic

The support logic consists of the video memory, the clocks, and the video DAC. The video memory consists of at least 256KB; its use and mapping depend on the mode selected.

Two clock sources provide the dot rate. The clock source is selected in the Miscellaneous Output register.

The video DAC contains the color palette that is used to convert the video data into the video signal sent to the display. Three analog signals (red, green, and blue) are output from the DAC.

The maximum number of colors displayed is 256 out of 256K, and the maximum number of gray shades is 64 out of 64.

VGA Components

The VGA function has four major functional areas: the CRT controller, the sequencer, the graphics controller, and the attribute controller.

CRT Controller

The CRT controller generates horizontal and vertical synchronization signal timings, addressing for the regenerative buffer, cursor and underline timings, and refresh addressing for the video memory.

Sequencer

The sequencer generates basic memory timings for the video memory and the character clock for controlling regenerative buffer fetches. It allows the system to access memory during active display intervals by periodically inserting dedicated system microprocessor memory cycles between the display memory cycles. Map mask registers in the sequencer are available to protect entire memory maps from being changed.

Graphics Controller

The graphics controller is the interface between the video memory and the attribute controller during active display times, and between video memory and the system microprocessor during memory accesses.

During active display times, memory data is latched and sent to the attribute controller. In graphics modes, the memory data is converted from parallel to serial bit-plane data before being sent; in alphanumeric modes, the parallel attribute data is sent.

During system accesses of video memory, the graphics controller can perform logical operations on the memory data before it reaches video memory or the system data bus. These logical operations are composed of four logical write modes and two logical read modes. The logical operators allow enhanced operations, such as a color compare in the read mode, individual bit masking during write modes, internal 32-bit writes in a single memory cycle, and writing to the display buffer on nonbyte boundaries.

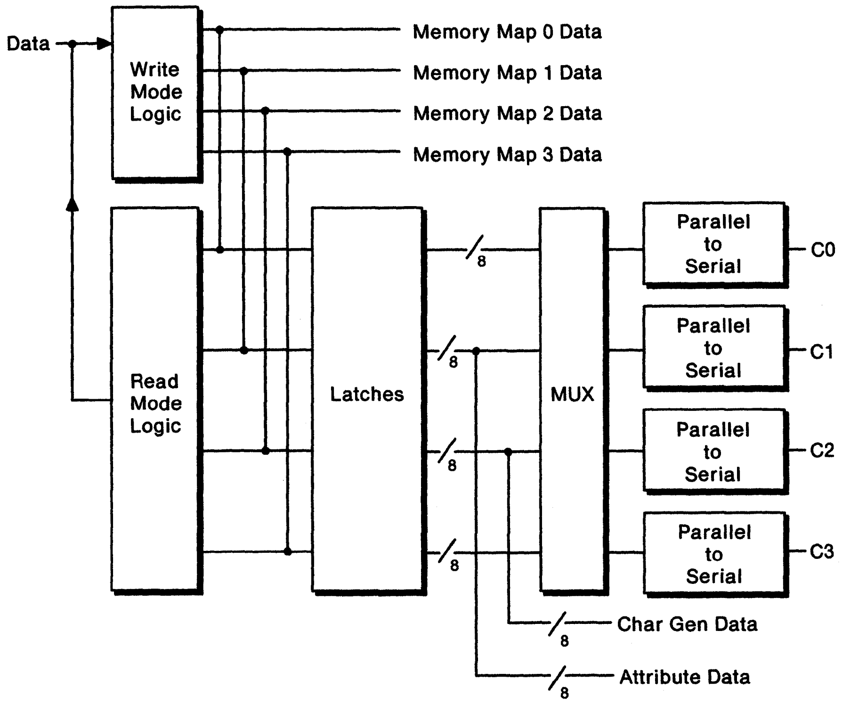


Figure 2-2. Graphics Controller

Attribute Controller

The attribute controller takes in data from video memory through the graphics controller and formats it for display. Attribute data in alphanumeric mode and serialized bit-plane data in graphics mode are converted to an 8-bit color value.

Each color value is selected from an internal color palette of 64 possible colors (except in 256-color mode). The color value is used as a pointer into the video DAC where it is converted to the analog signals that drive the display.

Blinking, underlining, cursor insertion, and PEL panning are also controlled in the attribute controller.

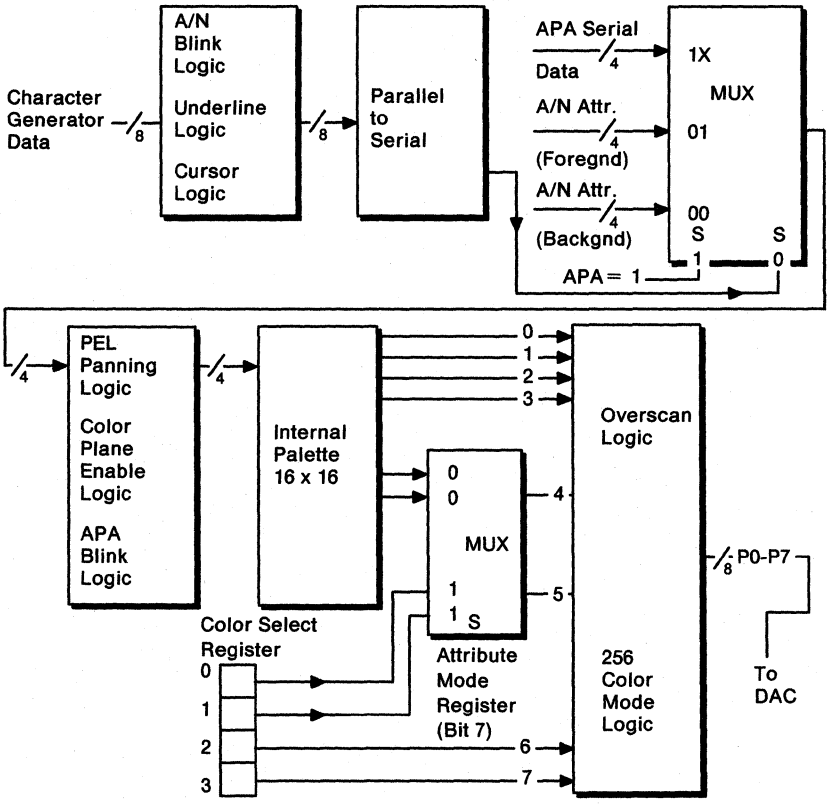


Figure 2-3. Attribute Controller

Hardware Considerations

The following are hardware characteristics of the Type 2 video subsystem that must be considered to ensure program compatibility with the Type 1 video subsystem.

Performance: Type 2 video generally runs faster than the Type 1 video subsystem; programs that depend on execution time of the video subsystem will operate differently.

Video Buffer Compatibility: For each of the video modes, the Type 2 video subsystem maintains a memory mapping that is the same as the Type 1. To maintain this compatibility, the internal addresses to video memory are manipulated so that video memory looks the same. When switching video modes, video data may not be at the same address in video memory.

BIOS calls to set and change modes make allowances for changes in addresses, and should be used for all mode switches.

Character Generator: Differences in the character generator for the Type 2 video subsystem increase the time it takes to load a new font. Because of the additional load time, there is a chance of briefly observing spurious data on the display. BIOS compensates for this during video mode sets.

Register Differences: The following bits for the Type 2 video subsystem differ from the Type 1:

- Bits 2 and 4 in the Clocking Mode register
- Bits 5 and 6 in the End Horizontal Blanking register
- Bits 2 and 4 in the Preset Row Scan register
- Bit 5 in the Address register of the attribute controller.

Modes of Operation

Certain modes on previous IBM display adapters distinguished between monochrome and color displays. For example, mode 0 was the same as mode 1 with the color burst turned off. Because color burst is not supported by the PS/2 video, the mode pairs are exactly the same. The support logic for the VGA function recognizes the type of display, and adjusts the output accordingly. When a monochrome display is attached, the colors for the color modes appear as shades of gray.

Mode 3+ is the default mode with a color display attached and mode 7+ is the default mode with a monochrome display attached.

The following figure describes the alphanumeric (A/N) and all points addressable (APA) graphics modes supported by BIOS. Each color is selected from 256K possibilities, and gray shades from 64 possibilities. The variations within the basic BIOS modes are selected through BIOS calls that set the number of scan lines. The scan line count is set before the mode call is made.

Mode (hex)	**Type**	**Colors**	**Alpha Format**	**Buffer Start**	**Box Size**	**Max. Pgs.**	**Freq.**	**Vert. PELs**
0,1	A/N	16	40 x 25	B8000	8 x 8	8	70 Hz	320 x 200
0*,1*	A/N	16	40 x 25	B8000	8 x 14	8	70 Hz	320 x 350
0+,1+	A/N	16	40 x 25	B8000	9 x 16	8	70 Hz	360 x 400
2,3	A/N	16	80 x 25	B8000	8 x 8	8	70 Hz	640 x 200
2*,3*	A/N	16	80 x 25	B8000	8 x 14	8	70 Hz	640 x 350
2+,3+	A/N	16	80 x 25	B8000	9 x 16	8	70 Hz	720 x 400
4,5	APA	4	40 x 25	B8000	8 x 8	1	70 Hz	320 x 200
6	APA	2	80 x 25	B8000	8 x 8	1	70 Hz	640 x 200
7	A/N	–	80 x 25	B8000	9 x 14	8	70 Hz	720 x 350
7+	A/N	–	80 x 25	B8000	9 x 16	8	70 Hz	720 x 400
D	APA	16	40 x 25	A0000	8 x 8	8	70 Hz	320 x 200
E	APA	16	80 x 25	A0000	8 x 8	4	70 Hz	640 x 200
F	APA	–	80 x 25	A0000	8 x 14	2	70 Hz	640 x 350
10	APA	16	80 x 25	A0000	8 x 14	2	70 Hz	640 x 350
11	APA	2	80 x 30	A0000	8 x 16	1	60 Hz	640 x 480
12	APA	16	80 x 30	A0000	8 x 16	1	60 Hz	640 x 480
13	APA	256	40 x 25	A0000	8 x 8	1	70 Hz	320 x 200

Note: * or + Enhanced modes

Figure 2-4. BIOS Video Modes

In the 200-scan-line modes, the data for each scan line is scanned twice. This double scanning allows the 200-scan-line image to be displayed in 400 scan lines.

Border support and double scanning depend on the mode selected. The following shows which modes use double scanning and which support a border.

Mode (Hex)	Double Scan	Border Support
0, 1	Yes	No
0*, 1*	No	No
0+, 1+	No	No
2, 3	Yes	Yes
2*, 3*	No	Yes
2+, 3+	No	Yes
4, 5	Yes	No
6	Yes	Yes
7	No	Yes
7+	No	Yes
D	Yes	No
E	Yes	Yes
F	No	Yes
10	No	Yes
11	No	Yes
12	No	Yes
13	Yes	Yes

Note: * or + Enhanced modes

Figure 2-5. Double Scanning and Border Support

Display Support

The video subsystem supports direct-drive analog displays. The displays must have a horizontal scan rate of 31.5 kHz, and a vertical scan rate capability of 50 to 70 Hz. Displays that use a digital input, such as the IBM Color Display, are *not* supported. The following figure summarizes the minimum display characteristics required to support VGA mode operation.

Parameter	Color	Monochrome
Horizontal Scan Rate	31.5 kHz	31.5 kHz
Vertical Scan Rate	50 to 70 Hz	50 to 70 Hz
Video Bandwidth	28 MHz	28 MHz
Maximum Horizontal Resolution	720 PELs	720 PELs
Maximum Vertical Resolution	480 Lines	480 Lines

Figure 2-6. Direct-Drive Analog Displays

Since color and monochrome displays run at the same scan rate, all modes work on both displays. The vertical gain of the display is controlled by the polarity of the vertical and horizontal synchronization pulses. This is done so 350, 400, or 480 lines can be displayed without adjusting the display. See "Signal Timing" on page 4-3 for more information.

Programmable Option Select

The video subsystem supports programmable option select (POS). The video subsystem is placed in the setup mode through the System Board Setup/Enable register (hex 0094). (For more information, see the system-specific sections.) For information on BIOS calls to enable or disable the video, see the *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference*.

Alphanumeric Modes

The alphanumeric modes are modes hex 0 through 3 and 7. The mode chart lists the variations of these modes (see Figure 2-4 on page 2-12). The data format for alphanumeric modes is the same as the data format on the IBM Color/Graphics Monitor Adapter, the IBM Monochrome Display Adapter, and the IBM Enhanced Graphics Adapter.

BIOS initializes the video subsystem according to the selected mode and loads the color values into the video DAC. These color values can be changed to give a different color set to select from. Bit 3 of the attribute byte can be redefined by the Character Map Select register to act as a switch between character sets, giving the programmer access to 512 characters at one time.

When an alphanumeric mode is selected, the BIOS transfers character font patterns from the ROM to map 2. The system stores the character data in map 0, and the attribute data in map 1. In the alphanumeric modes, the programmer views maps 0 and 1 as a single buffer. The CRT controller generates sequential addresses and fetches one character code byte and one attribute byte at a time. The character code and row scan count are combined to make up the address into map 2, which contains the character font. The appropriate dot patterns are then sent to the attribute controller, where color is assigned according to the attribute data.

Every display-character position in the alphanumeric mode is defined by two bytes in the display buffer. Both the color/graphics and the monochrome emulation modes use the following 2-byte character/attribute format.

Display Character Code Byte								Attribute Byte							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Even Address								Odd Address							

Figure 2-7. Character/Attribute Format

See "Characters and Keystrokes" for characters loaded during a BIOS mode set.

The functions of the attribute bytes are defined in the following table. Bit 7 can be redefined in the Attribute Mode Control register to give 16 possible background colors; its default is to control character blinking. Bit 3 can be redefined in the Character Map Select register to select between two character fonts; its default is to control foreground color selection.

Bit	Color	Function
7	B/I	Blinking or Background Intensity
6	R	Background Color
5	G	Background Color
4	B	Background Color
3	I/CS	Foreground Intensity or Character Font Select
2	R	Foreground Color
1	G	Foreground Color
0	B	Foreground Color

Figure 2-8. Attribute Byte Definitions

For more information about the attribute bytes, see "Character Map Select Register" on page 2-52 and "Attribute Mode Control Register" on page 2-91.

The following are the color values loaded by BIOS for the 16-color modes.

Intensity	Red	Green	Blue	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	White
1	0	0	0	Gray
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 (High Intensity)

Figure 2-9. BIOS Color Set

Both 40-column and 80-column alphanumeric modes are supported. The features of the 40-column alphanumeric modes (all variations of modes hex 0 and 1) are:

- 25 rows of 40 characters
- 2000 bytes of video memory per page
- One character byte and one attribute byte per character.

The features of the 80-column alphanumeric modes (all variations of modes hex 2, 3, and 7) are:

- 25 rows of 80 characters
- 4000 bytes of video memory per page
- One character byte and one attribute byte per character.

Graphics Modes

The graphics modes supported in BIOS are modes hex 4, 5, 6, F, and 11. The colors described in this section are generated when the BIOS is used to set the mode. BIOS initializes the video subsystem and the DAC palette to generate these colors. If the DAC palette is changed, different colors are generated.

320 x 200 Four-Color Graphics (Modes Hex 4 and 5)

Addressing, mapping, and data format are the same as the 320 x 200 PEL mode of the IBM Color/Graphics Monitor Adapter. The display buffer is configured at hex B8000. Bit image data is stored in memory maps 0 and 1. The two bit planes (C0 and C1) are each formed from bits from both memory maps.

Features of this mode are:

- A maximum of 200 rows of 320 PELs
- Double scanned to display as 400 rows
- Memory-mapped graphics
- Four colors for each PEL
- Four PELs per byte
- 16,000 bytes of read/write memory.

The video memory is organized into two banks of 8,000 bytes each, using the following format. Address hex B8000 contains the PEL information for the upper-left corner of the display area.

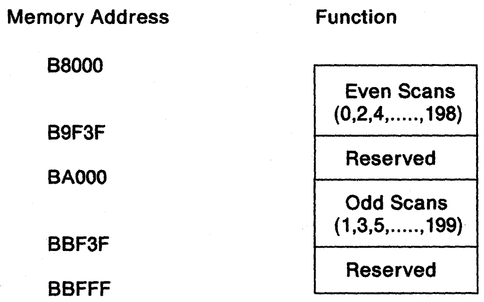


Figure 2-10. Video Memory Format

The following figure shows the format for each byte.

Bit	Function
7	C1 - First Display PEL
6	C0 - First Display PEL
5	C1 - Second Display PEL
4	C0 - Second Display PEL
3	C1 - Third Display PEL
2	C0 - Third Display PEL
1	C1 - Fourth Display PEL
0	C0 - Fourth Display PEL

Figure 2-11. PEL Format, Modes Hex 4 and 5

The color selected depends on the color set that is used. Color set 1 is the default. For information on changing the color set, see the *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference*.

Bits	Color Selected	
C1 C0	Color Set 1	Color Set 0
0 0	Black	Black
0 1	Light Cyan	Green
1 0	Light Magenta	Red
1 1	Intensified White	Brown

Figure 2-12. Color Selections, Modes Hex 4 and 5

640 x 200 Two-Color Graphics (Mode Hex 6)

Addressing, scan-line mapping, and data format are the same as the 640 x 200 PEL black and white mode of the IBM Color/Graphics Monitor Adapter. The display buffer is configured at hex B8000. Bit image data is stored in memory map 0 and comprises a single bit plane (C0). Features of this mode are:

- A maximum of 200 rows of 640 PELs
- Double scanned to display as 400 rows
- Same addressing and scan-line mapping as 320 x 200 graphics
- Two colors for each PEL
- Eight PELs per byte
- 16,000 bytes of read/write memory.

The following shows the format for each byte.

Bit	Function
7	First Display PEL
6	Second Display PEL
5	Third Display PEL
4	Fourth Display PEL
3	Fifth Display PEL
2	Sixth Display PEL
1	Seventh Display PEL
0	Eighth Display PEL

Figure 2-13. PEL Format, Mode Hex 6

The bit definition for each PEL is 0 equals black and 1 equals intensified white.

640 x 350 Graphics (Mode Hex F)

This mode emulates the EGA graphics with the monochrome display and the following attributes: black, video, blinking video, and intensified video. A resolution of 640 x 350 uses 56,000 bytes of video memory to support the four attributes. This mode uses maps 0 and 2; map 0 is the video bit plane (C0), and map 2 is the intensity bit plane (C2). Both planes reside at address hex A0000.

The two bits, one from each bit plane, define one PEL. The bit definitions are given in the following table.

C2	C0	PEL Color
0	0	Black
0	1	White
1	0	Blinking White
1	1	Intensified White

Figure 2-14. Bit Definitions C2,C0

Memory is organized with successive bytes defining successive PELs. The first eight PELs displayed are defined by the byte at hex A0000, the second eight PELs by the byte at hex A0001, and so on. The most-significant bit in each byte defines the first PEL for that byte.

Since both bit planes reside at address hex A0000, the user must select the plane to update through the Map Mask register of the sequence controller (see “Video Memory Organization” on page 2-24).

640 x 480 Two-Color Graphics (Mode Hex 11)

This mode provides two-color graphics with the same data format as mode 6. Addressing and mapping are shown under "Video Memory Organization" on page 2-24.

The bit image data is stored in map 0 and comprises a single bit plane (C0). The video buffer starts at hex A0000. The first byte contains the first eight PELs; the second byte, at hex A0001, contains the second eight PELs, and so on. The bit definition for each PEL is 0 equals black and 1 equals intensified white.

16-Color Graphics Modes (Modes Hex D, E, 10, and 12)

These modes support 16 colors. For all modes, the bit image data is stored in all four memory maps. Each memory map contains the data for one bit plane. The bit planes are C0 through C3 and represent the following colors:

C0 = Blue
C1 = Green
C2 = Red
C3 = Intensified

The four bits define each PEL on the screen by acting as an address (pointer) into the internal palette in the Extended Graphics mode.

The display buffer resides at address hex A0000. The Map Mask register selects any or all of the maps to be updated when the system writes to the display buffer.

256-Color Graphics Mode (Mode Hex 13)

This mode provides graphics with the capability of displaying 256 colors at one time.

The display buffer is sequential, starts at address hex A0000, and is 64,000 bytes long. The first byte contains the color information for the upper-left PEL. The second byte contains the second PEL, and so on, for 64,000 PELs (320 x 200). The bit image data is stored in all four memory maps and comprises four bit planes. The four bit planes are sampled twice to produce eight bit-plane values that address the video DAC.

In this mode, the internal palette of the video subsystem is loaded by BIOS and should not be changed. The first 16 locations in the external palette, which is in the video DAC, contain the colors compatible with the alphanumeric modes. The second 16 locations contain 16 evenly spaced gray shades. The next 216 locations contain values based on a hue-saturation-intensity model tuned to provide a usable, generic color set that covers a wide range of color values.

The following figure shows the color information that is compatible with the colors in other modes.

PEL Bits 7 6 5 4 3 2 1 0	Color Output
0 0 0 0 0 0 0 0	Black
0 0 0 0 0 0 0 1	Blue
0 0 0 0 0 0 1 0	Green
0 0 0 0 0 0 1 1	Cyan
0 0 0 0 0 1 0 0	Red
0 0 0 0 0 1 0 1	Magenta
0 0 0 0 0 1 1 0	Brown
0 0 0 0 0 1 1 1	White
0 0 0 0 1 0 0 0	Dark Gray
0 0 0 0 1 0 0 1	Light Blue
0 0 0 0 1 0 1 0	Light Green
0 0 0 0 1 0 1 1	Light Cyan
0 0 0 0 1 1 0 0	Light Red
0 0 0 0 1 1 0 1	Light Magenta
0 0 0 0 1 1 1 0	Yellow
0 0 0 0 1 1 1 1	Intensified White

Figure 2-15. Compatible Color Coding

Each color in the palette can be programmed to one of 256K different colors.

The features of this mode are:

- A maximum of 200 rows with 320 PELs
- Double scanned to display as 400 rows
- Memory-mapped graphics
- 256 of 256K colors for each PEL
- One byte per PEL
- 64,000 bytes of video memory.

Video Memory Organization

The display buffer consists of 256KB of dynamic read/write memory configured as four 64KB memory maps.

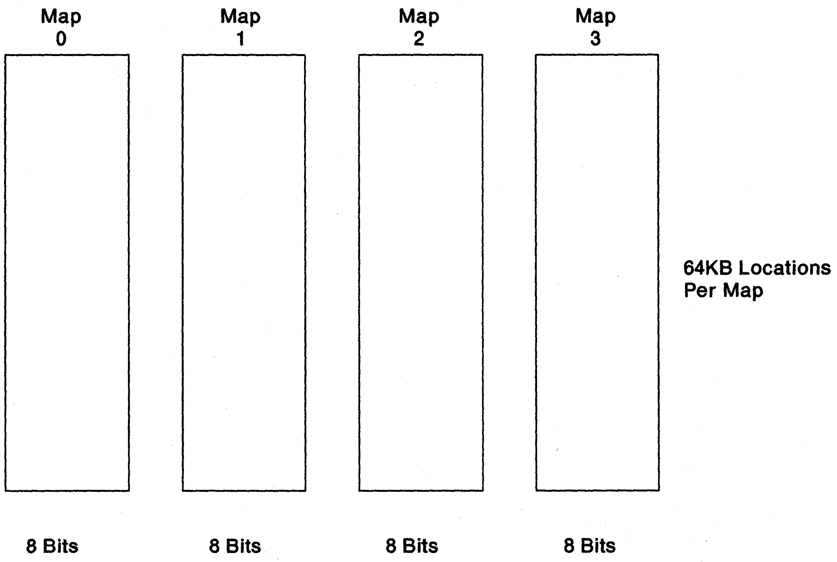


Figure 2-16. 256KB Video Memory Map

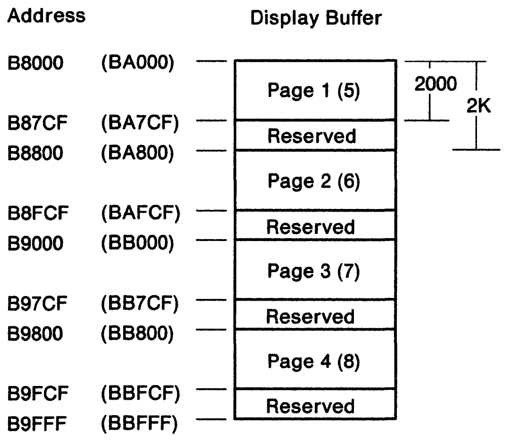
The starting address and size of the display buffer can be changed to maintain compatibility with other display adapters and application software. There are three configurations used by other adapters:

Address hex A0000 for a length of 64KB
Address hex B0000 for a length of 32KB
Address hex B8000 for a length of 32KB.

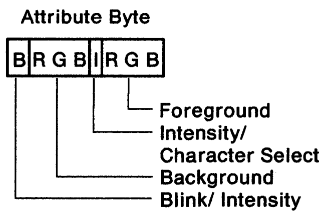
Memory Modes

The following pages show the memory organization for each of the BIOS modes.

Modes Hex 0, 1

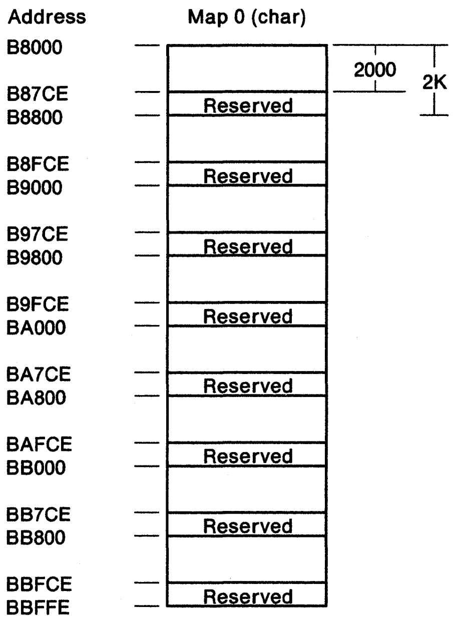


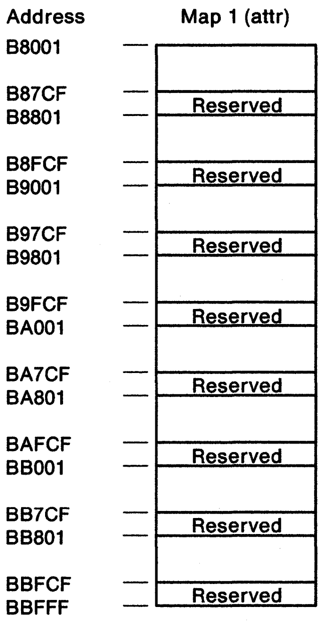
Storage Scheme



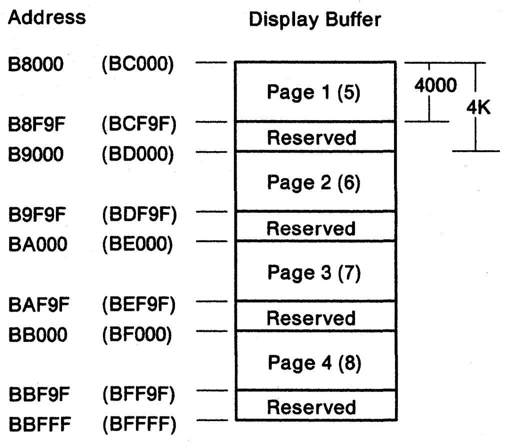
-16 colors per character

Character Byte
-Format is one character per byte

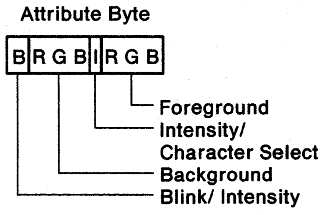




Modes Hex 2, 3

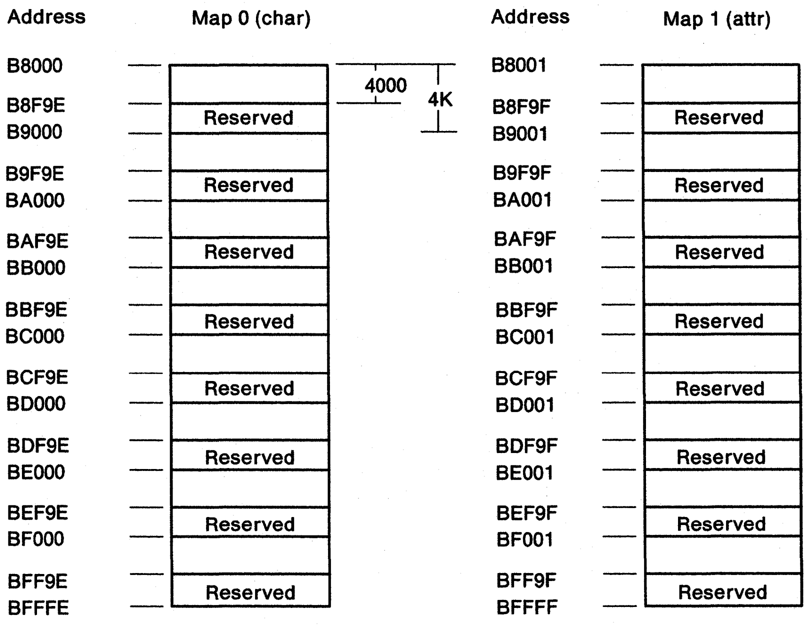


Storage Scheme

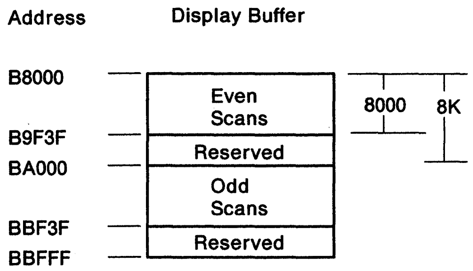


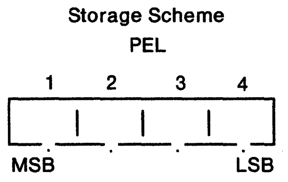
-16 colors per character

Character Byte
-Format is one character per byte

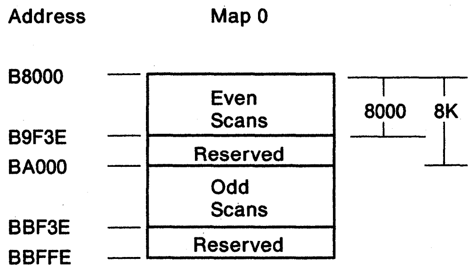


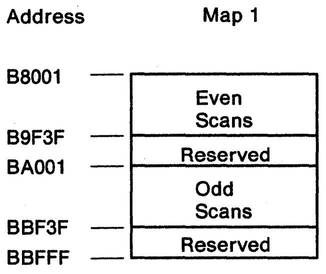
Modes Hex 4, 5



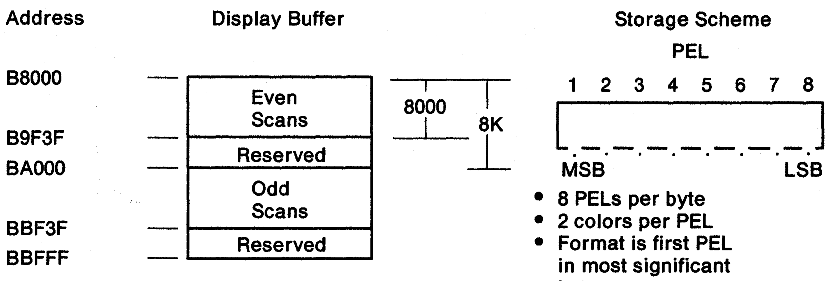


- 4 PELs per byte
- 4 colors per PEL
- Format is first PEL in 2 most significant bytes

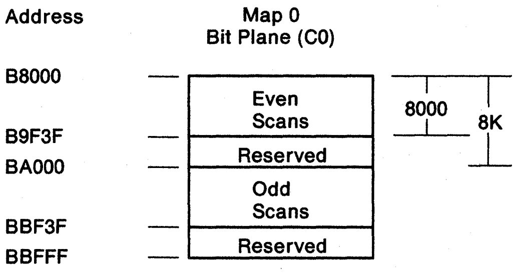




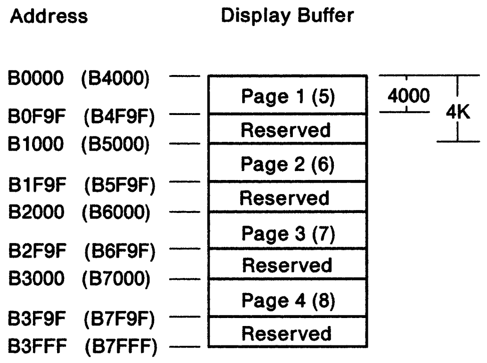
Mode Hex 6



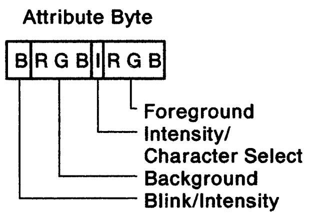
- 8 PELs per byte
- 2 colors per PEL
- Format is first PEL in most significant byte



Mode Hex 7

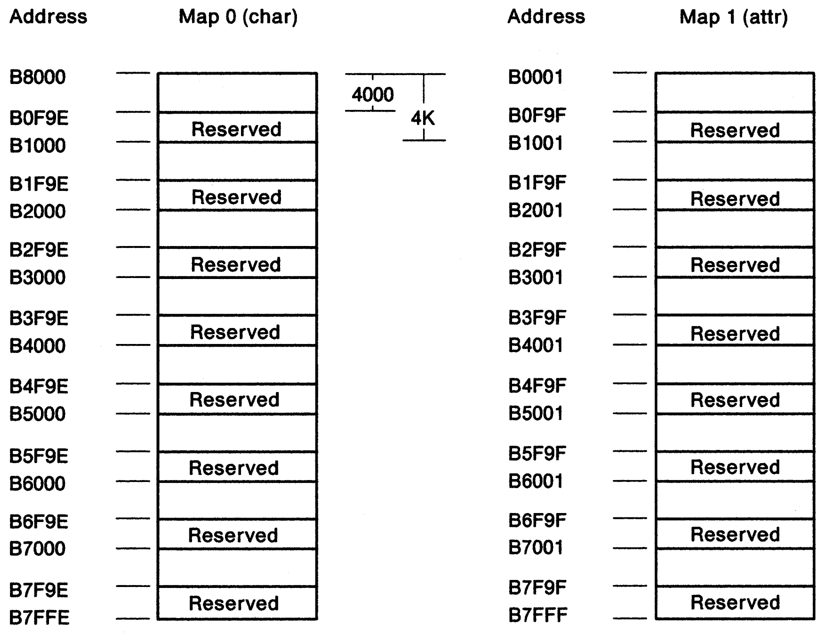


Storage Scheme

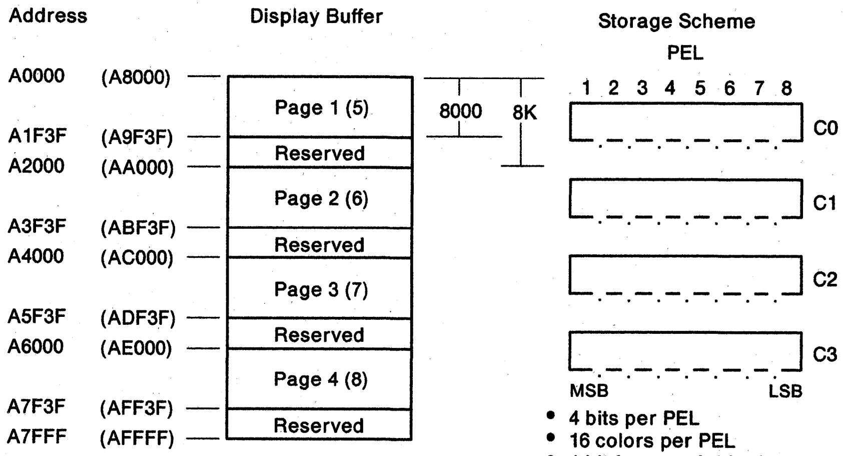


-Four attributes per character

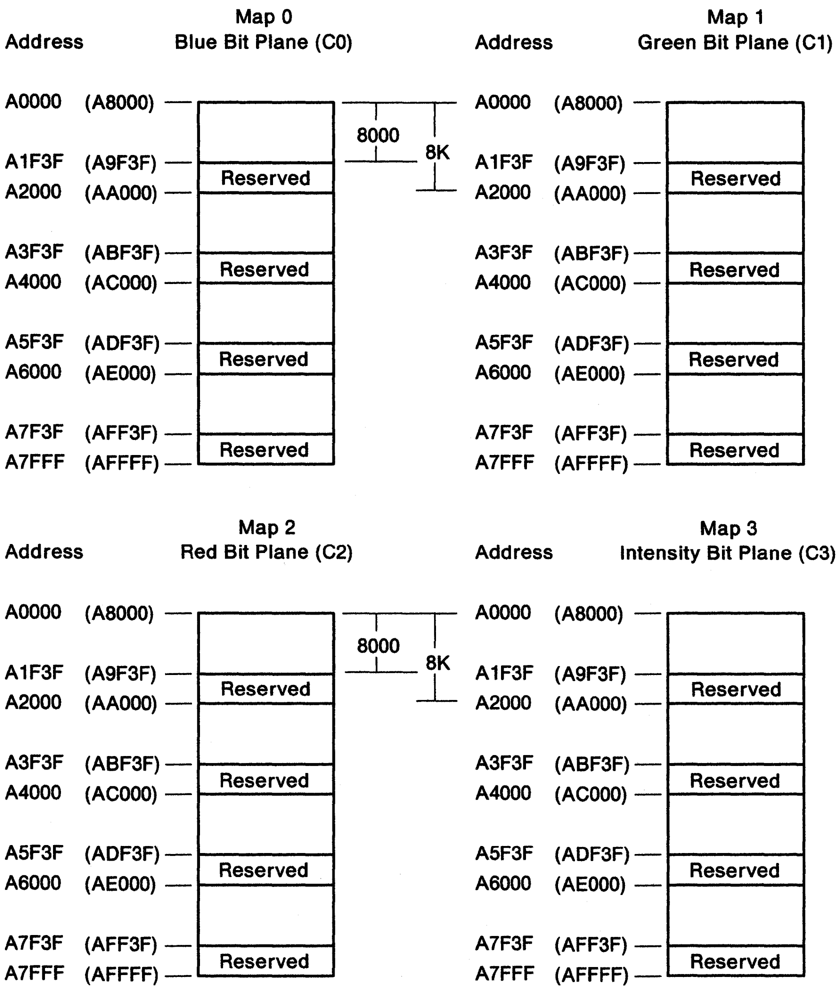
Character Byte
-Format is one character per byte.



Mode Hex D

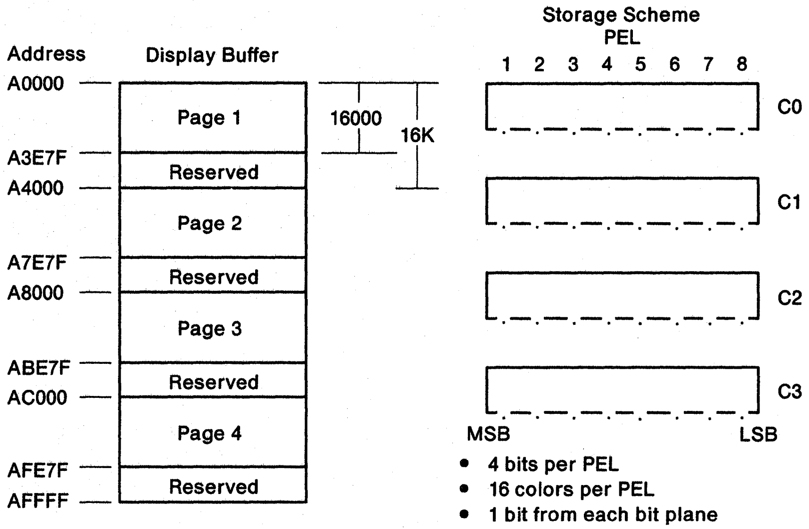


- 4 bits per PEL
- 16 colors per PEL
- 1 bit from each bit plane (C3,C2,C1,CO) per PEL
- Format is first PEL in most significant byte of all 4 bit planes

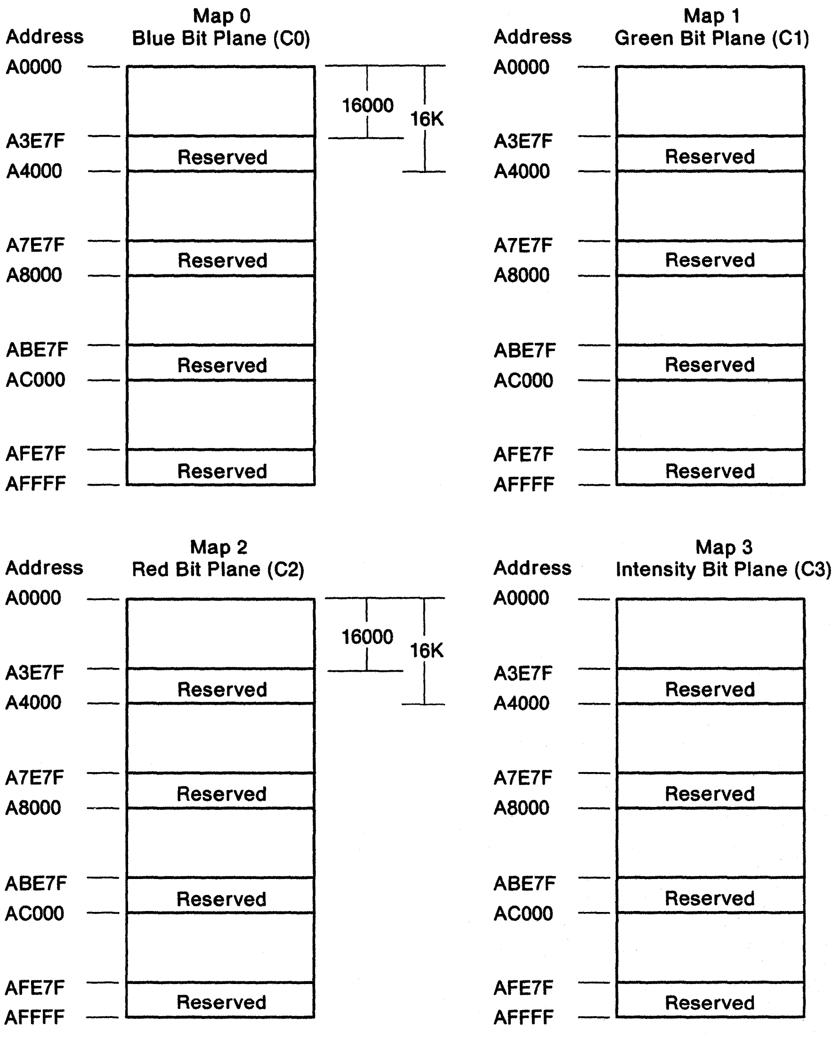


Map 0
Address
Blue Bit Plane (C0)
A0000 (A8000)
A1F3F (A9F3F)
A2000 (AA000)
A3F3F (ABF3F)
A4000 (AC000)
A5F3F (ADF3F)
A6000 (AE000)
A7F3F (AFF3F)
A7FFF (AFFFF)
Reserved
Reserved
Reserved
Reserved
8000
8K
Map 1
Address
Green Bit Plane (C1)
A0000 (A8000)
A1F3F (A9F3F)
A2000 (AA000)
A3F3F (ABF3F)
A4000 (AC000)
A5F3F (ADF3F)
A6000 (AE000)
A7F3F (AFF3F)
A7FFF (AFFFF)
Reserved
Reserved
Reserved
Reserved
Map 2
Address
Red Bit Plane (C2)
A0000 (A8000)
A1F3F (A9F3F)
A2000 (AA000)
A3F3F (ABF3F)
A4000 (AC000)
A5F3F (ADF3F)
A6000 (AE000)
A7F3F (AFF3F)
A7FFF (AFFFF)
Reserved
Reserved
Reserved
Reserved
8000
8K
Map 3
Address
Intensity Bit Plane (C3)
A0000 (A8000)
A1F3F (A9F3F)
A2000 (AA000)
A3F3F (ABF3F)
A4000 (AC000)
A5F3F (ADF3F)
A6000 (AE000)
A7F3F (AFF3F)
A7FFF (AFFFF)
Reserved
Reserved
Reserved
Reserved

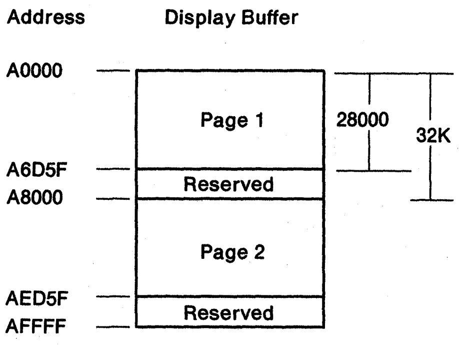
Mode Hex E

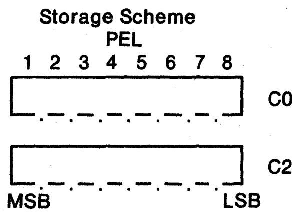


- 4 bits per PEL
- 16 colors per PEL
- 1 bit from each bit plane (C3,C2,C1,C0) per PEL
- Format is first PEL in most significant byte of all 4 bit planes

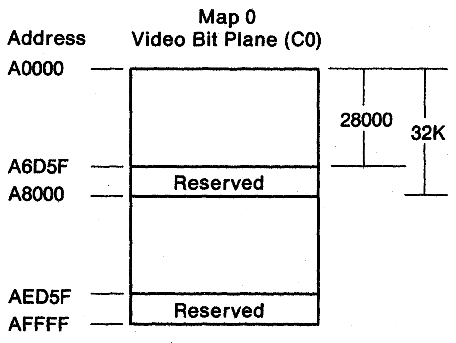

Map 0
Blue Bit Plane (C0)
Map 1
Green Bit Plane (C1)
Map 2
Red Bit Plane (C2)
Map 3
Intensity Bit Plane (C3)
Address
A0000
A3E7F
A4000
A7E7F
A8000
ABE7F
AC000
AFE7F
AFFFF
Reserved
16000
16K

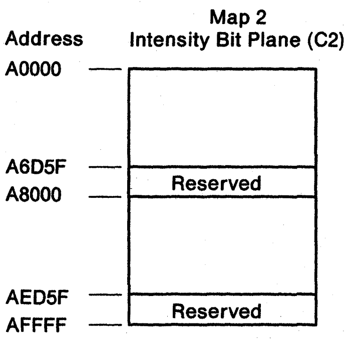
Mode Hex F



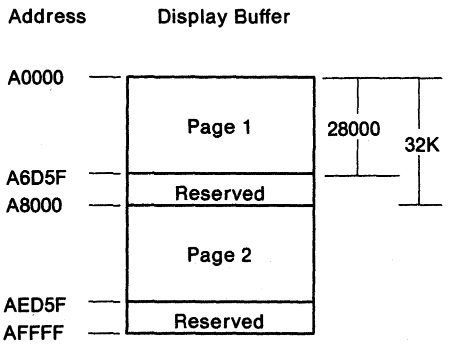


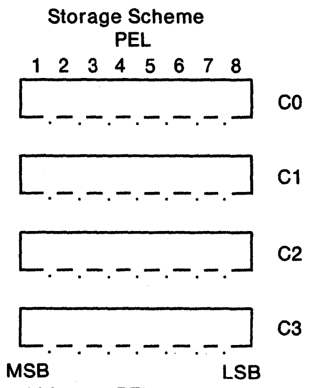
- 2 bits per PEL
- 4 attributes per PEL
- 1 bit from each bit plane (C2,C0)
- Format is first PEL in most significant byte of video and intensity bit planes



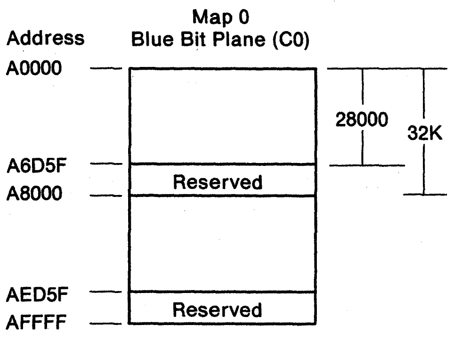


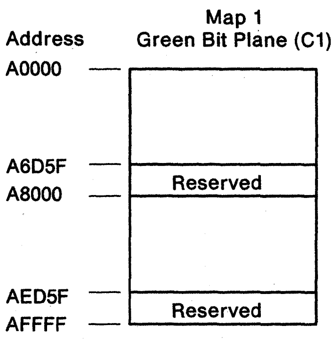
Mode Hex 10

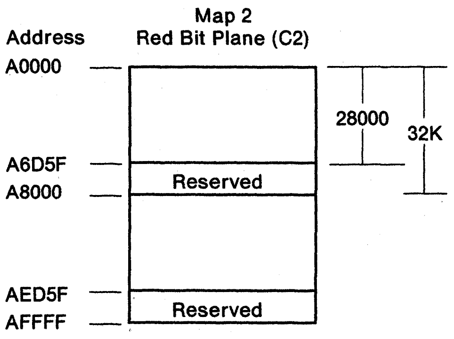


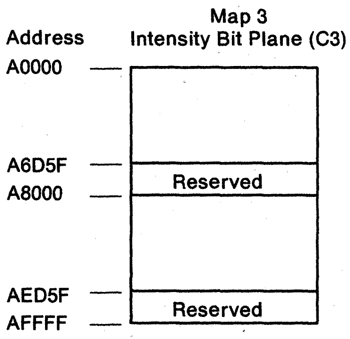


- 4 bits per PEL
- 16 colors per PEL
- 1 bit from each bit plane (C3,C2,C1,C0) per PEL
- Format is first PEL in most significant byte of all 4 bit planes

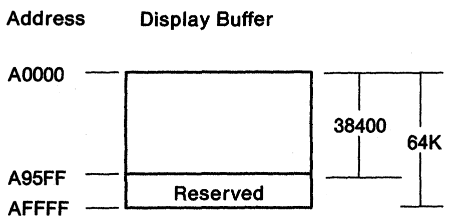


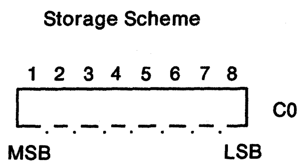




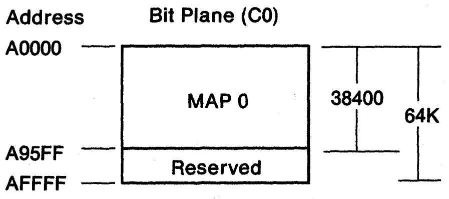


Mode Hex 11

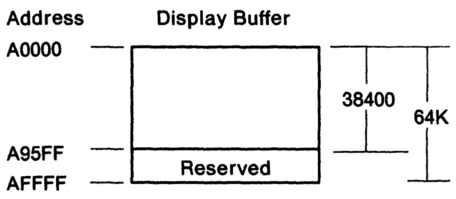


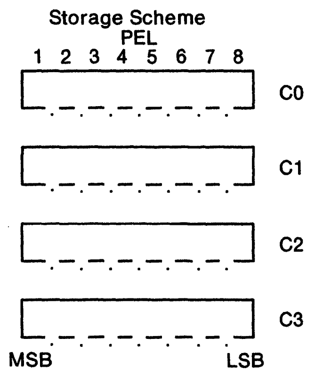


- One bit per PEL
- Two attributes per PEL
- Format is first PEL in most significant byte

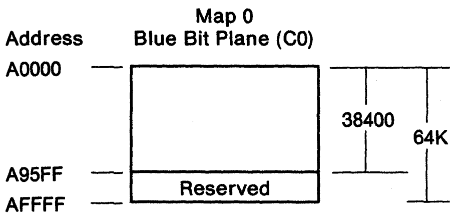


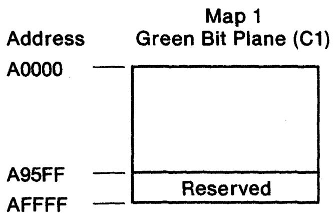
Mode Hex 12

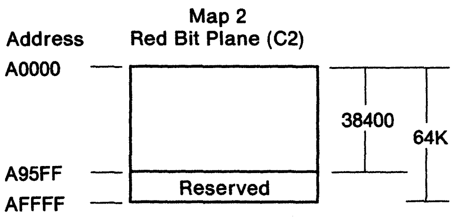


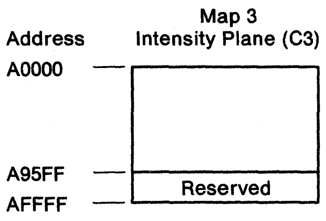


- 4 bits per PEL
- 16 colors per PEL
- 1 bit from each bit plane (C3,C2,C1,C0) per PEL
- Format is first PEL in most significant byte of all 4 bit planes

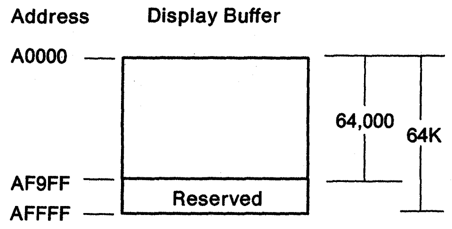




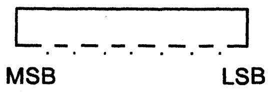




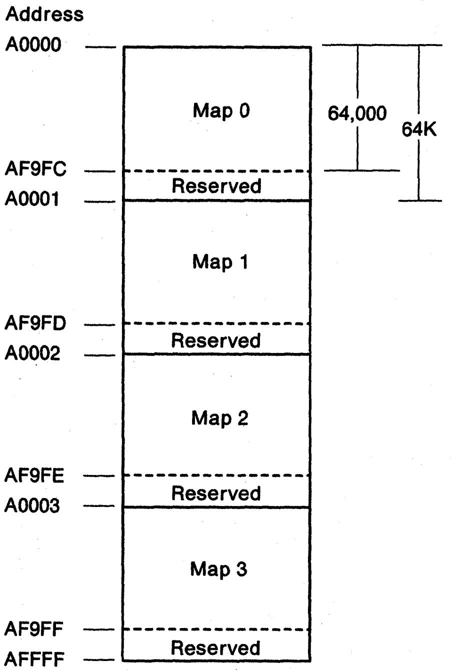
Mode Hex 13



Storage Scheme



- 8 bits per PEL
- 256 colors per PEL
- 1 PEL per byte
- First PEL is at address A0000



Memory Operations

Memory operations consist of write and read operations.

Write Operations

When the system is writing to the display buffer, the maps are enabled by the logical decode of the memory address and the Map Mask register. The addresses used for video memory depend on the mode selected. The data flow for a system write operation is illustrated in the following figure.

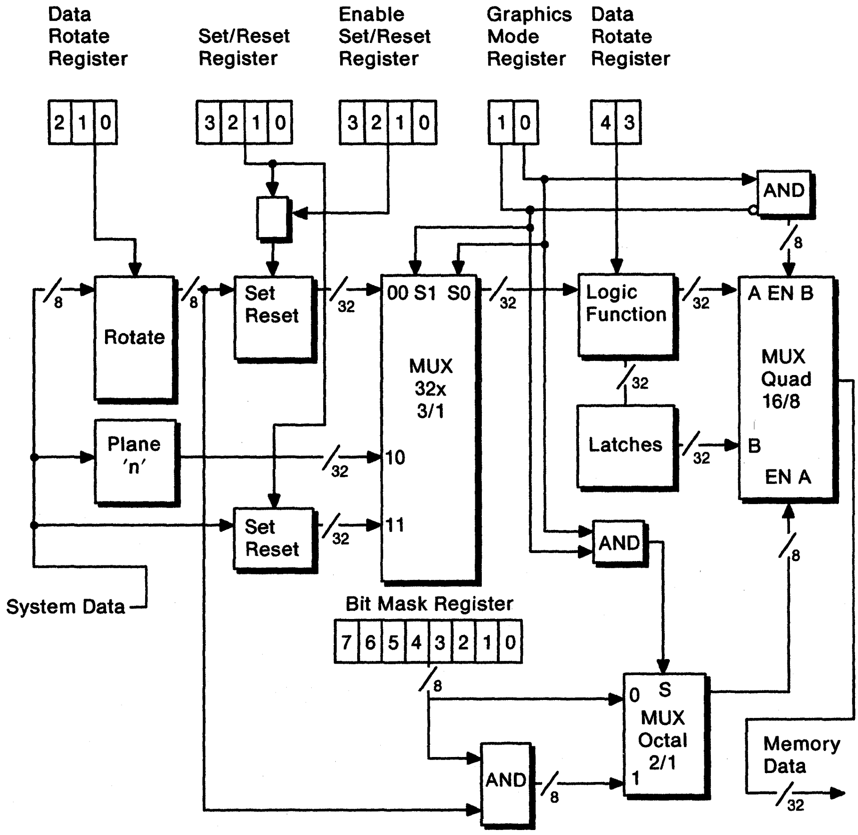


Figure 2-17. Data Flow for Write Operations

Read Operations

The two ways to read the video buffer are selected through the Graphics Mode register in the graphics controller. The mode 0 read operation returns the 8-bit value determined by the logical decode of the memory address and, if applicable, the Read Map Select register. The mode 1 read operation returns the 8-bit value resulting from the color compare operation controlled by the Color Compare and Color Don't Care registers. The data flow for the color compare operation is shown in the following figure.

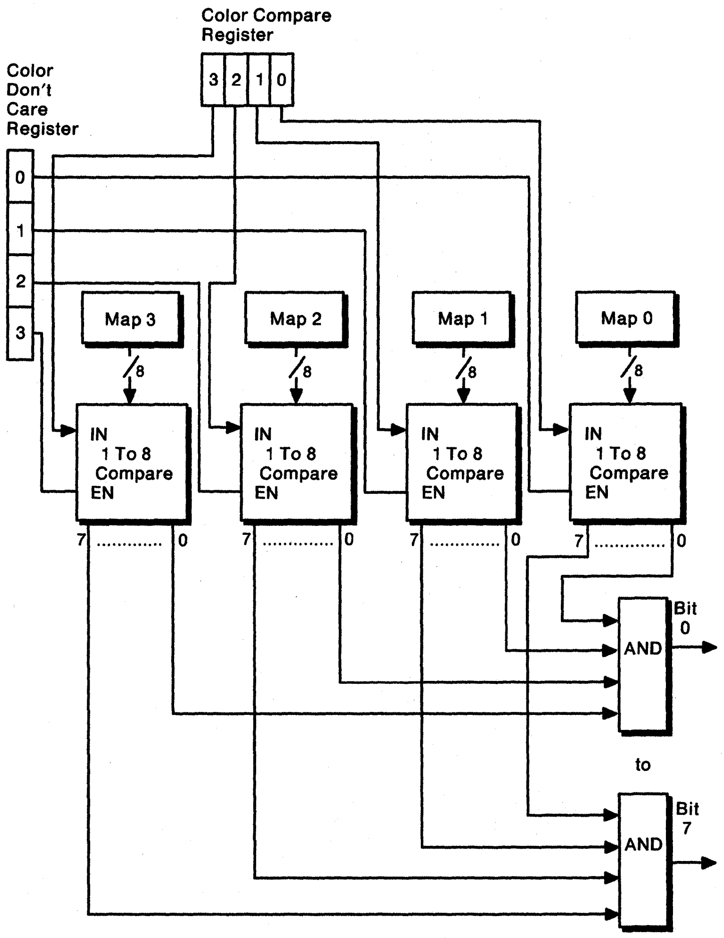


Figure 2-18. Color Compare Operations

Registers

There are six groups of registers in the video subsystem. All video registers are readable except the system data latches and the attribute address flip-flop. The following figure lists the register groups, their I/O addresses with the type of access (read or write), and page reference numbers.

The question mark in the address can be a hex B or D depending on the setting of the I/O address bit in the Miscellaneous Output register, described in "General Registers" on page 2-42.

Note: All registers in the video subsystem are read/write. The value of reserved bits in these registers must be preserved. Read the register first and change only the bits required.

Registers	R/W	Port Address	Page Reference
General Registers			2-42
Sequencer Registers			2-47
Address Register	R/W	03C4	
Data Registers	R/W	03C5	
CRT Controller Registers			2-55
Address Register	R/W	03?4	
Data Registers	R/W	03?5	
Graphics Controller Registers			2-78
Address Register	R/W	03CE	
Data Registers	R/W	03CF	
Attribute Controller Registers			2-89
Address Register	R/W	03C0	
Data Registers	W	03C0	
	R	03C1	
Video DAC Palette Registers			2-103
Write Address	R/W	03C8	
Read Address	W	03C7	
Data	R/W	03C9	
PEL Mask	R/W	03C6	

Figure 2-19. Video Subsystem Register Overview

General Registers

Register	Read Address	Write Address
Miscellaneous Output Register	03CC	03C2
Input Status Register 0	03C2	–
Input Status Register 1	03?A	–
Feature Control Register	03CA	03?A
Video Subsystem Enable Register	03C3	03C3

Figure 2-20. General Registers

Miscellaneous Output Register

The read address for this register is hex 03CC and its write address is hex 03C2.

7	6	5	4	3	2	1	0
VSP	HSP	–	–	CS		ERAM	IOS

–	:	Set to 0, Undefined on Read
VSP	:	Vertical Sync Polarity
HSP	:	Horizontal Sync Polarity
CS	:	Clock Select
ERAM	:	Enable RAM
IOS	:	I/O Address Select

Figure 2-21. Miscellaneous Output Register, Hex 03CC/03C2

The register fields are defined as follows:

VSP When set to 0, the Vertical Sync Polarity field (bit 7) selects a positive 'vertical retrace' signal. This bit works with bit 6 to determine the vertical size.

HSP When set to 0, the Horizontal Sync Polarity field (bit 6) selects a positive 'horizontal retrace' signal. Bits 7 and 6 select the vertical size as shown in the following figure.

Bits 7 6	**Vertical Size**
0 0	Reserved
0 1	400 lines
1 0	350 lines
1 1	480 lines

Figure 2-22. Display Vertical Size

CS The Clock Select field (bits 3, 2) selects the clock source according to the following figure. The external clock is driven through the auxiliary video extension. The input clock should be kept between 14.3 MHz and 28.4 MHz.

CS Field (binary)	**Function**
0 0	Selects 25.175 MHz clock for 640/320 Horizontal PELs
0 1	Selects 28.322 MHz clock for 720/360 Horizontal PELs
1 0	Selects External Clock
1 1	Reserved

Figure 2-23. Clock Select Definitions

ERAM When set to 0, the Enable RAM field (bit 1) disables address decode for the display buffer from the system.

IOS The I/O Address Select field (bit 0) selects the CRT controller addresses. When set to 0, this bit sets the CRT controller addresses to hex 03Bx and the address for the Input Status Register 1 to hex 03BA for compatibility with the monochrome adapter. When set to 1, this bit sets CRT controller addresses to hex 03Dx and the Input Status Register 1 address to hex 03DA for compatibility with the color/graphics adapter. The write addresses to the Feature Control register are affected in the same manner.

Input Status Register 0

The address for this read-only register is address hex 03C2. *Do not write to* this register.

7	6	5	4	3	2	1	0
CI	-	-	SS	-	-	-	-

```
 - : Undefined on Read
CI : CRT Interrupt
SS : Switch Sense
```

Figure 2-24. Input Status Register 0, Hex 03C2

The register fields are defined as follows:

CI When the CRT Interrupt field (bit 7) is 1, a vertical retrace interrupt is pending.

SS BIOS uses the Switch Sense field (bit 4) in determining the type of display attached.

Input Status Register 1

The address for this read-only register is address hex 03DA or 03BA. *Do not write to* this register.

7	6	5	4	3	2	1	0
–	–	–	–	VR	–	–	DE

–	:	Undefined on Read
VR	:	Vertical Retrace
DE	:	Display Enable

Figure 2-25. Input Status Register 1, Hex 03DA/03BA

The register fields are defined as follows:

VR When the Vertical Retrace field (bit 3) is 1, it indicates a vertical retrace interval. This bit can be programmed, through the Vertical Retrace End register, to generate an interrupt at the start of the vertical retrace.

DE When the Display Enable field (bit 0) is 1, it indicates a horizontal or vertical retrace interval. This bit is the real-time status of the inverted 'display enable' signal. In the past, programs have used this status bit to restrict screen updates to the inactive display intervals to reduce screen flicker. The video subsystem is designed to eliminate this software requirement; screen updates may be made at any time without screen degradation.

Feature Control Register

The write address of this register is hex 03DA or 03BA; its read address is hex 03CA. All bits are reserved.

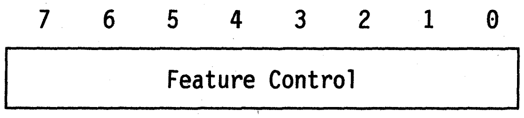


Figure 2-26. Feature Control Register, Hex 03DA/03BA and 03CA

Video Subsystem Enable Register

This register (hex 03C3) is reserved. To disable address decoding by the video subsystem, use BIOS INT 10 call, AH = hex 12, BL = hex 32.

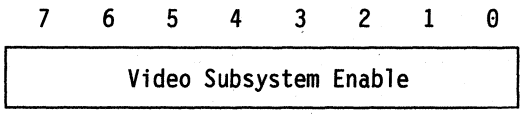


Figure 2-27. Video Subsystem Enable Register, Hex 03C3

Sequencer Registers

The Address register is at address hex 03C4 and the data registers are at address hex 03C5. All registers within the sequencer are read/write.

Register	Index (Hex)
Sequencer Address	–
Reset	00
Clocking Mode	01
Map Mask	02
Character Map Select	03
Memory Mode	04

Figure 2-28. Sequencer Registers

Sequencer Address Register

The Address register is at address hex 03C4. This register is loaded with an index value that points to the desired sequencer data register.

7	6	5	4	3	2	1	0
–	–	–	–	–	SA		

```
 – : Set to 0, Undefined on Read
SA : Sequencer Address
```

Figure 2-29. Sequencer Address Register

The register field is defined as follows:

SA The Sequencer Address field (bits 2 – 0) contains the index value that points to the data register to be accessed.

Reset Register

This read/write register has an index of hex 00; its address is hex 03C5.

7	6	5	4	3	2	1	0
–	–	–	–	–	–	SR	ASR

– :	Set to 0, Undefined on Read
SR :	Synchronous Reset
ASR :	Asynchronous Reset

Figure 2-30. Reset Register, Index Hex 00

The register fields are defined as follows:

SR When set to 0, the Synchronous Reset field (bit 1) commands the sequencer to synchronously clear and halt. Bits 1 and 0 must be 1 to allow the sequencer to operate. To prevent the loss of data, bit 1 must be set to 0 during the active display interval before changing the clock selection. The clock is changed through the Clocking Mode register or the Miscellaneous Output register.

ASR When set to 0, the Asynchronous Reset field (bit 0) commands the sequencer to asynchronously clear and halt. Resetting the sequencer with this bit can cause loss of video data.

Clocking Mode Register

This read/write register has an index of hex 01; its address is hex 03C5.

7	6	5	4	3	2	1	0
–	–	SO	SH4	DC	SL	1	D89

–	:	Set to 0, Undefined on Read
1	:	Set to 1, Undefined on Read
SO	:	Screen Off
SH4	:	Shift 4
DC	:	Dot Clock
SL	:	Shift Load
D89	:	8/9 Dot Clocks

Figure 2-31. Clocking Mode Register, Index Hex 01

The register fields are defined as follows:

SO When set to 1, the Screen Off field (bit 5) turns off the display and assigns maximum memory bandwidth to the system. Although the display is blanked, the synchronization pulses are maintained. This bit can be used for rapid full-screen updates.

SH4 When the Shift 4 field (bit 4) and Shift Load field (bit 2) are set to 0, the video serializers are loaded every character clock. When the Shift 4 field is set to 1, the video serializers are loaded every fourth character clock, which is useful when 32 bits are fetched per cycle and chained together in the shift registers.

Extended Graphics mode behaves as if this bit is set to 0; therefore, programs should set it to 0.

DC When set to 0, the Dot Clock field (bit 3) selects the normal dot clocks derived from the sequencer master clock input. When set to 1, the master clock is divided by 2 to generate the dot clock. All other timings are affected because they are derived from the dot clock. The dot clock divided by 2 is used for 320 and 360 horizontal PEL modes.

SL When the Shift Load field (bit 2) and Shift 4 field (bit 4) are set to 0, the video serializers are loaded every character clock. When the Shift Load field (bit 2) is set to 1, the video serializers are loaded every other character clock, which is useful when 16 bits are fetched per cycle and chained together in the shift registers.

Extended Graphics mode behaves as if this bit is set to 0; therefore, programs should set it to 0.

D89 When set to 0, the 8/9 Dot Clocks field (bit 0) directs the sequencer to generate character clocks 9 dots wide; when set to 1, it directs the sequencer to generate character clocks 8 dots wide. The 9-dot mode is for alphanumeric modes 0+, 1+, 2+, 3+, 7, and 7+ only; the 9th dot equals the 8th dot for ASCII codes hex C0 through DF. All other modes must use 8 dots per character clock. (See the Enable Line Graphics Character Code field in the Attribute Mode Control register on page 2-91.)

Map Mask Register

This read/write register has an index of hex 02; its address is hex 03C5.

7	6	5	4	3	2	1	0
–	–	–	–	M3E	M2E	M1E	M0E

–	: Set to 0, Undefined on Read
M3E	: Map 3 Enable
M2E	: Map 2 Enable
M1E	: Map 1 Enable
M0E	: Map 0 Enable

Figure 2-32. Map Mask Register, Index Hex 02

The register fields are defined as follows:

M3E, M2E, M1E, M0E

When set to 1, the map enable fields (bits 3, 2, 1, and 0) enable system access to the corresponding map. If all maps are enabled (chain 4 mode), the system can write its 8-bit value to all four maps in a single memory cycle. This substantially reduces the system overhead during display updates in graphics modes.

Data scrolling operations can be enhanced by enabling all maps and writing the display buffer address with the data stored in the system data latches. This is a read-modify-write operation.

When access to odd or even maps (odd/even modes) are selected, maps 0 and 1 and maps 2 and 3 should have the same map mask value.

When chain 4 mode is selected, all maps should be enabled.

Character Map Select Register

This register has an index of hex 03; its address is hex 03C5. In alphanumeric modes, bit 3 of the attribute byte normally defines the foreground intensity. This bit can be redefined as a switch between character sets, allowing 512 displayable characters. To enable this feature:

1. Set the extended memory bit in the Memory Mode register (index hex 04) to 1.
2. Select different values for character map A and character map B.

This function is supported by BIOS and is a function call within the character generator routines.

7	6	5	4	3	2	1	0
–	–	MAH	MBH	MAL		MBL	

–	:	Set to 0, Undefined on Read
MAH	:	Character Map A Select (MSB)
MBH	:	Character Map B Select (MSB)
MAL	:	Character Map A Select (LS bits)
MBL	:	Character Map B Select (LS bits)

Figure 2-33. Character Map Select Register, Index Hex 03

The register fields are defined as follows:

MAH The Character Map A Select field (bit 5) is the most-significant bit for selecting the location of character map A.

MBH The Character Map B Select field (bit 4) is the most-significant bit for selecting the location of character map B.

MAL The Character Map A Select field (bits 3, 2) and Character Map A Select field (bit 5) select the location of character map A. Map A is the area of map 2 containing the character font table used to generate characters when attribute bit 3 is set to 1. The selection is shown in the following figure.

Bits 5 3 2	Map Selected	Table Location
0 0 0	0	1st 8KB of Map 2
0 0 1	1	3rd 8KB of Map 2
0 1 0	2	5th 8KB of Map 2
0 1 1	3	7th 8KB of Map 2
1 0 0	4	2nd 8KB of Map 2
1 0 1	5	4th 8KB of Map 2
1 1 0	6	6th 8KB of Map 2
1 1 1	7	8th 8KB of Map 2

Figure 2-34. Character Map Select A

MBL The Character Map B Select field (bits 1, 0) and Character Map B Select field (bit 4) select the location of character map B. Map B is the area of map 2 containing the character font table used to generate characters when attribute bit 3 is set to 0. The selection is shown in the following figure.

Bits 4 1 0	Map Selected	Table Location
0 0 0	0	1st 8KB of Map 2
0 0 1	1	3rd 8KB of Map 2
0 1 0	2	5th 8KB of Map 2
0 1 1	3	7th 8KB of Map 2
1 0 0	4	2nd 8KB of Map 2
1 0 1	5	4th 8KB of Map 2
1 1 0	6	6th 8KB of Map 2
1 1 1	7	8th 8KB of Map 2

Figure 2-35. Character Map Select B

Memory Mode Register

This register has an index of hex 04; its address is hex 03C5.

7	6	5	4	3	2	1	0
–	–	–	–	CH4	OE	EM	–

```
  - : Set to 0, Undefined on Read
CH4 : Chain 4
 OE : Odd/Even
 EM : Extended Memory
```

Figure 2-36. Memory Mode Register, Index Hex 04

The register fields are defined as follows:

CH4 The Chain 4 field (bit 3) controls the map selected during system read operations. When set to 0, this bit enables system addresses to sequentially access data within a bit map by using the Map Mask register. When set to 1, this bit causes the 2 low-order bits to select the map accessed as shown in the following figure.

Address Bits A1 A0	Map Selected
0 0	0
0 1	1
1 0	2
1 1	3

Figure 2-37. Map Selection, Chain 4

OE When the Odd/Even field (bit 2) is set to 0, even system addresses access maps 0 and 2, while odd system addresses access maps 1 and 3. When set to 1, system addresses sequentially access data within a bit map, and the maps are accessed according to the value in the Map Mask register (hex 02).

EM When set to 1, the Extended Memory field (bit 1) enables the video memory from 64KB to 256KB. This bit must be set to 1 to enable the character map selection described for the previous register.

CRT Controller Registers

A data register is accessed by writing its index to the Address register at address hex 03D4 or 03B4, and then writing the data to the access port at address hex 03D5 or 03B5. The I/O address used depends on the setting of the I/O address select bit (bit 0) in the Miscellaneous Output register, which is described in "General Registers" on page 2-42. The following figure shows the variable part of the address as a question mark.

Note: When modifying a register, the setting of reserved bits must be preserved. Read the register first and change only the bits required.

Register Name	**Address (Hex)**	**Index (Hex)**
Address	03?4	–
Horizontal Total	03?5	00
Horizontal Display-Enable End	03?5	01
Start Horizontal Blanking	03?5	02
End Horizontal Blanking	03?5	03
Start Horizontal Retrace Pulse	03?5	04
End Horizontal Retrace	03?5	05
Vertical Total	03?5	06
Overflow	03?5	07
Preset Row Scan	03?5	08
Maximum Scan Line	03?5	09
Cursor Start	03?5	0A
Cursor End	03?5	0B
Start Address High	03?5	0C
Start Address Low	03?5	0D
Cursor Location High	03?5	0E
Cursor Location Low	03?5	0F
Vertical Retrace Start	03?5	10
Vertical Retrace End	03?5	11
Vertical Display-Enable End	03?5	12
Offset	03?5	13
Underline Location	03?5	14
Start Vertical Blanking	03?5	15
End Vertical Blanking	03?5	16
CRT Mode Control	03?5	17
Line Compare	03?5	18
Index values not listed are reserved.		

Figure 2-38. CRT Controller Registers

Address Register

This register is at address hex 03B4 or 03D4, and is loaded with an index value that points to the data registers within the CRT controller.

7	6	5	4	3	2	1	0
–	–	–	Index				

– : Set to 0, Undefined on Read

Figure 2-39. CRT Controller Address Register, Hex 03B4/03D4

The register field is defined as follows:

Index This field (bits 4 – 0) is the index that points to the data register accessed through address hex 03D5 or 03B5.

Horizontal Total Register

This register has an index of hex 00; its address is hex 03D5 or 03B5.

The Horizontal Total register (bits 7 – 0) defines the total number of characters in the horizontal scan interval including the retrace time. The value directly controls the period of the 'horizontal retrace' signal. A horizontal character counter in the CRT controller counts the character clock inputs; comparators are used to compare the register value with the horizontal width of the character to provide horizontal timings. All horizontal and vertical timings are based on this register.

7	6	5	4	3	2	1	0
Horizontal Total							

Figure 2-40. Horizontal Total Register, Index Hex 00

The value contained in the register is the total number of characters minus 5.

Horizontal Display-Enable End Register

This register has an index of hex 01; its address is hex 03D5 or 03B5.

```
  7   6   5   4   3   2   1   0
┌────────────────────────────────┐
│  Horizontal Display Enable End │
└────────────────────────────────┘
```

Figure 2-41. Horizontal Display Enable-End Register, Index Hex 01

The Horizontal Display-Enable End register (bits 7 – 0) defines the length of the 'horizontal display-enable' signal and determines the number of character positions per horizontal line. The value in this register is the total number of displayed characters minus 1.

Start Horizontal Blanking Register

This register has an index of hex 02; its address is hex 03D5 or 03B5.

```
  7   6   5   4   3   2   1   0
┌────────────────────────────────┐
│    Start Horizontal Blanking   │
└────────────────────────────────┘
```

Figure 2-42. Start Horizontal Blanking Register, Index Hex 02

The value in the Start Horizontal Blanking register (bits 7 – 0) is the horizontal character count at which the 'horizontal blanking' signal goes active.

End Horizontal Blanking Register

This register has an index of hex 03; its address is hex 03D5 or 03B5. It determines when the 'horizontal blanking' signal will go active.

7	6	5	4	3	2	1	0
1	DES		EB				

```
  1 : Set to 1, Undefined on Read
DES : Display Enable Skew Control
 EB : End Blanking
```

Figure 2-43. End Horizontal Blanking Register, Index Hex 03

The register fields are defined as follows:

DES The Display Enable Skew Control field (bits 6, 5) determines the amount of skew of the 'display enable' signal. This skew control is needed to provide sufficient time for the CRT controller to read a character and attribute code from the video buffer, gain access to the character generator, and go through the Horizontal PEL Panning register in the attribute controller. Each access requires the 'display enable' signal to be skewed one character clock so that the video output is synchronized with the horizontal and vertical retrace signals. The skew values are shown in the following figure.

DES Field (binary)	Amount of Skew
0 0	No character clock skew
0 1	One character clock skew
1 0	Two character clock skew
1 1	Three character clock skew

Figure 2-44. Display Enable Skew

Note: Character skew is not adjustable on the Type 2 video and the bits are ignored; however, programs should set these bits for the appropriate skew to maintain compatibility.

EB The End Blanking field (bits 4 – 0) contains the 5 low-order bits of a 6-bit value that is compared with the value in the Start Horizontal Blanking register to determine when the 'horizontal blanking' signal goes inactive. The most-significant bit is bit 7 in the End Horizontal Retrace register (index hex 05).

To program these bits for a signal width of W, the following algorithm is used: the width W, in character clock units, is added to the value from the Start Horizontal Blanking register. The 6 low-order bits of the result are the 6-bit value programmed.

Start Horizontal Retrace Pulse Register

This register has an index of hex 04; its address is hex 03D5 or 03B5.

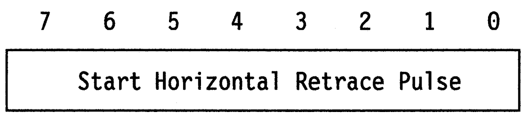


Figure 2-45. Start Horizontal Retrace Pulse Register, Index Hex 04

The Start Horizontal Retrace Pulse register (bits 7 – 0) is used to center the screen horizontally by specifying the character position where the 'horizontal retrace' signal goes active.

End Horizontal Retrace Register

This register has an index of hex 05; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
EB5	HRD		EHR				

EB5	:	End Horizontal Blanking, Bit 5
HRD	:	Horizontal Retrace Delay
EHR	:	End Horizontal Retrace

Figure 2-46. End Horizontal Retrace Register, Index Hex 05

The register fields are defined as follows:

EB5 The End Horizontal Blanking, Bit 5 field (bit 7) is the most-significant bit of the end horizontal blanking value in the End Horizontal Blanking register (index hex 03).

HRD The Horizontal Retrace Delay field (bits 6, 5) controls the skew of the 'horizontal retrace' signal. The value of this field is the amount of skew provided (from 0 to 3 character clock units). For certain modes, the 'horizontal retrace' signal takes up the entire blanking interval. Some internal timings are generated by the falling edge of the 'horizontal retrace' signal. To ensure that the signals are latched properly, the 'retrace' signal is started before the end of the 'display enable' signal and then skewed several character clock times to provide the proper screen centering.

EHR The End Horizontal Retrace field (bits 4—0) is compared with the Start Horizontal Retrace register to give a horizontal character count at which the 'horizontal retrace' signal goes inactive.

To program these bits with a signal width of W, the following algorithm is used: the width W, in character clock units, is added to the value in the Start Retrace register. The 5 low-order bits of the result are the 5-bit value programmed.

Vertical Total Register

This register has an index of hex 06; its address is hex 03D5 or 03B5.

```
  7    6    5    4    3    2    1    0
+---------------------------------------+
|            Vertical Total             |
+---------------------------------------+
```

Figure 2-47. Vertical Total Register, Index Hex 06

The Vertical Total register (bits 7 – 0) contains the 8 low-order bits of a 10-bit vertical total. The value for the vertical total is the number of horizontal raster scans on the display, including vertical retrace, minus 2. This value determines the period of the ‘vertical retrace’ signal.

Bits 8 and 9 are in the Overflow register (index hex 07).

Overflow Register

This register has an index of hex 07; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
VRS9	VDE9	VT9	LC8	VBS8	VRS8	VDE8	VT8

VRS9	:	Vertical Retrace Start, Bit 9
VDE9	:	Vertical Display Enable End, Bit 9
VT9	:	Vertical Total, Bit 9
LC8	:	Line Compare, Bit 8
VBS8	:	Vertical Blanking Start, Bit 8
VRS8	:	Vertical Retrace Start, Bit 8
VDE8	:	Vertical Display Enable End, Bit 8
VT8	:	Vertical Total, Bit 8

Figure 2-48. CRT Overflow Register, Index Hex 07

The register fields are defined as follows:

VRS9	This is bit 9 of the Vertical Retrace Start register (index hex 10).
VDE9	This is bit 9 of the Vertical Display-Enable End register (index hex 12).
VT9	This is bit 9 of the Vertical Total register (index hex 06).
LC8	This is bit 8 of the Line Compare register (index hex 18).
VBS8	This is bit 8 of the Start Vertical Blanking register (index hex 15).
VRS8	This is bit 8 of the Vertical Retrace Start register (index hex 10).
VDE8	This is bit 8 of the Vertical Display-Enable End register (index hex 12).
VT8	This is bit 8 of the Vertical Total register (index hex 06).

Preset Row Scan Register

This register has an index of hex 08; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
–	BP		SRS				

–	Set to 0, Undefined on Read
BP	Byte Panning
SRS	Starting Row Scan Count

Figure 2-49. Preset Row Scan Register, Index Hex 08

The register fields are defined as follows:

BP The Byte Panning field (bits 6, 5) controls byte panning in multiple shift modes. (BIOS modes do not use multiple shift operation.) These bits are used in PEL-panning operations, and should normally be set to 0.

Extended Graphics mode behaves as if these bits are set to 0; therefore, programs should set it to 0.

SRS The Starting Row Scan Count field (bits 4–0) specifies the row scan count for the row starting after a vertical retrace. The row scan counter is incremented every horizontal retrace time until the maximum row scan occurs. When the maximum row scan is reached, the row scan counter is cleared (not preset).

Note: The CRT controller latches the start address at the start of the vertical retrace. These register values should be loaded during the active display time.

Maximum Scan Line Register

This register has an index of hex 09; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
DSC	LC9	VBS9	MSL				

DSC	200 to 400 Line Conversion (Double Scanning)
LC9	Line Compare, Bit 9
VBS9	Start Vertical Blanking, Bit 9
MSL	Maximum Scan Line

Figure 2-50. Maximum Scan Line Register, Index Hex 09

The register fields are defined as follows:

DSC When the 200 to 400 Line Conversion field (bit 7) is set to 1, 200-scan-line video data is converted to 400-scan-line output. To do this, the clock in the row scan counter is divided by 2, which allows the 200-line modes to be displayed as 400 lines on the display (this is called double scanning; each line is displayed twice). When set to 0, the clock to the row scan counter is equal to the horizontal scan rate.

LC9 The Line Compare, Bit 9 field (bit 6) is bit 9 of the Line Compare register (index hex 18).

VBS9 The Start Vertical Blanking, Bit 9 field (bit 5) is bit 9 of the Start Vertical Blanking register (index hex 15).

MSL The Maximum Scan Line field (bits 4 – 0) specifies the number of scan lines per character row. The value of this field is the maximum row scan number minus 1.

Cursor Start Register

This register has an index of hex 0A; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
–	–	CO	RSCB				

```
   – : Set to 0, Undefined on Read
  CO : Cursor Off
RSCB : Row Scan Cursor Begins
```

Figure 2-51. Cursor Start Register, Index Hex 0A

The register fields are defined as follows:

CO When the Cursor Off field (bit 5) is set to 1, the cursor is disabled.

RSCB The Row Scan Cursor Begins field (bits 4–0) specifies the row within the character box where the cursor begins. The value of this field is the first line of the cursor minus 1. When this value is greater than that in the Cursor End register, no cursor is displayed.

Cursor End Register

This register has an index of hex 0B; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
–	CSK		RSCE				

```
   – : Set to 0, Undefined on Read
 CSK : Cursor Skew Control
RSCE : Row Scan Cursor Ends
```

Figure 2-52. Cursor End Register, Index Hex 0B

The register fields are defined as follows:

CSK The Cursor Skew Control field (bits 6, 5) controls the skew of the cursor. The skew value delays the cursor by the selected number of character clocks from 0 to 3. For example, a skew of 1 moves the cursor right one position on the screen.

RSCE The Row Scan Cursor Ends field (bits 4–0) specifies the row within the character box where the cursor ends. If this value is less than that in the Cursor Start register, no cursor is displayed.

Start Address High Register

This register has an index of hex 0C; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
Start Address High							

Figure 2-53. Start Address High Register, Index Hex 0C

The Start Address High register (bits 7 – 0) contains the 8 high-order bits of a 16-bit value that specifies the starting address for the regenerative buffer. The start address points to the first address after the vertical retrace on each screen refresh.

Note: The CRT controller latches the start address at the start of the vertical retrace. These register values should be loaded during the active display time.

Start Address Low Register

This register has an index of hex 0D; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
Start Address Low							

Figure 2-54. Start Address Low Register, Index Hex 0D

The Start Address Low register (bits 7 – 0) contains the 8 low-order bits of the starting address for the regenerative buffer.

Cursor Location High Register

This register has an index of hex 0E; its address is hex 03D5 or 03B5.

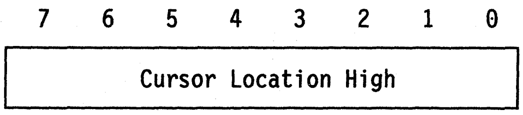


Figure 2-55. Cursor Location High Register, Index Hex 0E

The Cursor Location High register (bits 7 – 0) contains the 8 high-order bits of the 16-bit cursor location.

Cursor Location Low Register

This register has an index of hex 0F; its address is hex 03D5 or 03B5.

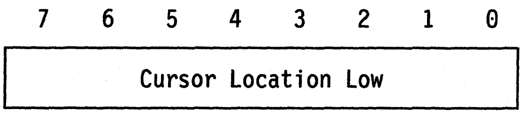


Figure 2-56. Cursor Location Low Register, Index Hex 0F

The Cursor Location Low register (bits 7 – 0) contains the 8 low-order bits of the cursor location.

Vertical Retrace Start Register

This register has an index of hex 10; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
Vertical Retrace Start							

Figure 2-57. Vertical Retrace Start Register, Index Hex 10

The Vertical Retrace Start register (bits 7 – 0) contains the 8 low-order bits of the 9-bit start position for the 'vertical retrace' signal; it is programmed in horizontal scan lines. Bit 8 is in the Overflow register (index hex 07).

Vertical Retrace End Register

This register has an index of hex 11; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
PR	S5R	EVI	CVI	VRE			

PR	:	Protect Registers 0-7
S5R	:	Select 5 Refresh Cycles
EVI	:	Enable Vertical Interrupt
CVI	:	Clear Vertical Interrupt
VRE	:	Vertical Retrace End

Figure 2-58. Vertical Retrace End Register, Index Hex 11

The register fields are defined as follows:

PR	When the Protect Registers 0 – 7 field (bit 7) is set to 1, write access to the CRT controller registers at index 00 through 07 is disabled. The line compare bit in the Overflow register (index hex 07) is not protected.
S5R	When the Select 5 Refresh Cycles field (bit 6) is set to 1, five memory refresh cycles per horizontal line are generated. When set to 0, three refresh cycles are selected. Selecting five refresh cycles allows use of the VGA chip with 15.75 kHz displays. This bit should be set to 0 for supported operations. It is set to 0 by a BIOS mode set, a reset, or a power on.
EVI	When the Enable Vertical Interrupt field (bit 5) is set to 0, it enables a vertical retrace interrupt. The vertical retrace interrupt is IRQ2. This interrupt level can be shared, therefore, to determine whether the video generated the interrupt, check the CRT interrupt bit in Input Status Register 0.
CVI	When the Clear Vertical Interrupt field (bit 4) is set to 0, it clears a vertical retrace interrupt. At the end of the active vertical display time, a flip-flop is set to indicate an interrupt. An interrupt handler resets this flip-flop by first setting this bit to 0, then resetting it to 1.
VRE	The Vertical Retrace Start register is compared with the 4 bits in the Vertical Retrace End field (bits 3 – 0) to determine where the 'vertical retrace' signal goes inactive. It is programmed in units of horizontal scan lines. To program these bits with a signal width of W, the following algorithm is used: the width W, in horizontal scan units, is added to the value in the Start Vertical Retrace register. The 4 low-order bits of the result are the 4-bit value programmed.

Vertical Display-Enable End Register

This register has an index of hex 12; its address is hex 03D5 or 03B5.

```
  7    6    5    4    3    2    1    0
+---------------------------------------+
|      Vertical Display Enable End      |
+---------------------------------------+
```

Figure 2-59. Vertical Display-Enable End Register, Index Hex 12

The Vertical Display-Enable End register (bits 7 – 0) contains the 8 low-order bits of a 10-bit value that defines the vertical-display-enable end position. The 2 high-order bits are contained in the Overflow register (index hex 07). The 10-bit value is equal to the total number of scan lines minus 1.

Offset Register

This register has an index of hex 13; its address is hex 03D5 or 03B5.

```
  7    6    5    4    3    2    1    0
+---------------------------------------+
|                Offset                 |
+---------------------------------------+
```

Figure 2-60. Offset Register, Index Hex 13

The Offset register (bits 7 – 0) specifies the logical line width of the screen. The starting memory address for the next character row is larger than the current character row by 2 or 4 times the value of these bits. Depending on the method of clocking the CRT controller, this address is either a word or doubleword address.

Underline Location Register

This register has an index of hex 14; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
–	DW	CB4	SUL				

–	: Set to 0, Undefined on Read
DW	: Doubleword Mode
CB4	: Count By 4
SUL	: Start Underline

Figure 2-61. Underline Location Register, Index Hex 14

The register fields are defined as follows:

DW When the Doubleword Mode field (bit 6) is set to 1, memory addresses are doubleword addresses. See the description of the word/byte mode bit (bit 6) in the CRT Mode Control register on page 2-74.

CB4 When the Count By 4 field (bit 5) is set to 1, the memory-address counter is clocked with the character clock divided by 4, which is used when doubleword addresses are used.

SUL The Start Underline field (bits 4 – 0) specifies the horizontal scan line of a character row on which an underline occurs. The value programmed is the scan line desired minus 1.

Start Vertical Blanking Register

This register has an index of hex 15; its address is hex 03D5 or 03B5.

```
  7   6   5   4   3   2   1   0
┌───────────────────────────────┐
│    Start Vertical Blanking    │
└───────────────────────────────┘
```

Figure 2-62. Start Vertical Blanking Register, Index Hex 15

The Start Vertical Blanking register (bits 7 – 0) contains the 8 low-order bits of a 10-bit value that specifies the starting location for the 'vertical blanking' signal. Bit 8 is in the Overflow register (index hex 07) and bit 9 is in the Maximum Scan Line register (index hex 09). The 10-bit value is the horizontal scan line count at which the 'vertical blanking' signal becomes active minus 1.

End Vertical Blanking Register

This register has an index of hex 16; its address is hex 03D5 or 03B5.

```
  7   6   5   4   3   2   1   0
┌───────────────────────────────┐
│     End Vertical Blanking     │
└───────────────────────────────┘
```

Figure 2-63. End Vertical Blanking Register, Index Hex 16

The End Vertical Blanking register (bits 7 – 0) specifies the horizontal scan count at which the 'vertical blanking' signal becomes inactive. The register is programmed in units of the horizontal scan line.

To program these bits with a 'vertical blanking' signal of width W, the following algorithm is used: the width W, in horizontal scan line units, is added to the value in the Start Vertical Blanking register minus 1. The 8 low-order bits of the result are the 8-bit value programmed.

CRT Mode Control Register

This register has an index of hex 17; its address is hex 03D5 or 03B5.

7	6	5	4	3	2	1	0
RST	WB	ADW	–	CB2	HRS	SRC	CMS0

–	:	Set to 0, Undefined on Read
RST	:	Hardware Reset
WB	:	Word/Byte Mode
ADW	:	Address Wrap
CB2	:	Count By Two
HRS	:	Horizontal Retrace Select
SRC	:	Select Row Scan Counter
CMS0	:	CMS 0

Figure 2-64. CRT Mode Control Register, Index Hex 17

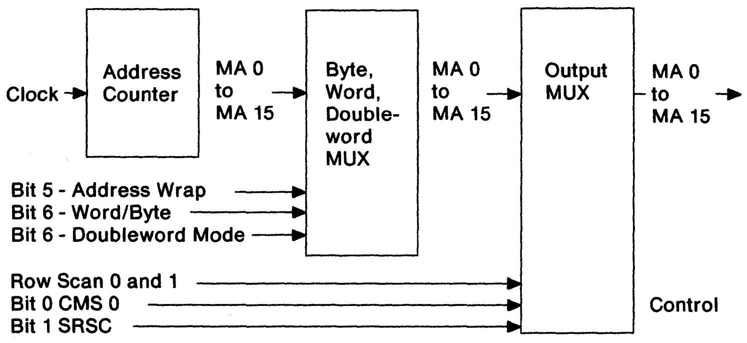
The register fields are defined as follows:

RST When the Hardware Reset field (bit 7) is set to 0, this bit disables the horizontal and vertical retrace signals and forces them to an inactive level. When set to 1, this bit enables the horizontal and vertical retrace signals. This bit does not reset any other registers or signal outputs.

WB When the Word/Byte Mode field (bit 6) is set to 0, the word mode is selected. The word mode shifts the memory-address counter bits down 1 bit; the most-significant bit of the counter appears on the least-significant bit of the memory address outputs.

The doubleword bit in the Underline Location register (index hex 14) also controls the addressing. When the doubleword bit is 0, the word/byte bit selects the mode. When the doubleword bit is set to 1, the addressing is shifted by 2 bits.

When the Word/Byte Mode field is set to 1, it selects the byte address mode. See the following figure for address output details.



Memory Address Outputs	Modes of Addressing		
	Byte	**Word**	**Doubleword**
MA 0	MA 0	MA 15 or 13	MA 12
MA 1	MA 1	MA 0	MA 13
MA 2	MA 2	MA 1	MA 0
MA 3	MA 3	MA 2	MA 1
MA 4	MA 4	MA 3	MA 2
MA 5	MA 5	MA 4	MA 3
MA 6	MA 6	MA 5	MA 4
MA 7	MA 7	MA 6	MA 5
MA 8	MA 8	MA 7	MA 6
MA 9	MA 9	MA 8	MA 7
MA 10	MA 10	MA 9	MA 8
MA 11	MA 11	MA 10	MA 9
MA 12	MA 12	MA 11	MA 10
MA 13	MA 13	MA 12	MA 11
MA 14	MA 14	MA 13	MA 12
MA 15	MA 15	MA 14	MA 13

Figure 2-65. CRT Memory Address Mapping

ADW The Address Wrap field (bit 5) selects the memory-address bit, bit MA 13 or MA 15, that appears on the output pin MA 0 in the word address mode. If VGA is not in the word address mode, bit 0 from the address counter appears on the output pin, MA 0.

When set to 1, the Address Wrap field bit selects MA 15. In odd/even mode, this bit should be set to 1 because 256KB of video memory is installed on the system board. (Bit MA 13 is selected in applications where only 64KB is present. This function maintains compatibility with the IBM Color/Graphics Monitor Adapter.)

CB2	When the Count By Two field (bit 3) is set to 0, the address counter uses the character clock. When set to 1, the address counter uses the character clock input divided by 2. This bit is used to create either a byte or word refresh address for the display buffer.
HRS	The Horizontal Retrace Select field (bit 2) selects the clock that controls the vertical timing counter. The clocking is either the horizontal retrace clock or horizontal retrace clock divided by 2. When set to 1, the horizontal retrace clock is divided by 2. Dividing the clock effectively doubles the vertical resolution of the CRT controller. The vertical counter has a maximum resolution of 1024 scan lines because the vertical total value is 10 bits wide. If the vertical counter is clocked with the horizontal retrace divided by 2, the vertical resolution is doubled to 2048 scan lines.
SRC	The Select Row Scan Counter field (bit 1) selects the source of bit 14 of the output multiplexer. When set to 0, bit 1 of the row scan counter is the source. When set to 1, bit 14 of the address counter is the source.
CMS0	The CMS 0 field (bit 0) selects the source of bit 13 of the output multiplexer. When set to 0, bit 0 of the row scan counter is the source. When set to 1, bit 13 of the address counter is the source. The CRT controller used on the IBM Color/Graphics Adapter was capable of using 128 horizontal scan-line addresses. For VGA to obtain 640-by-200 graphics resolution, the CRT controller is programmed for 100 horizontal scan lines with two scan-line addresses per character row. Row scan address bit 0 becomes the most-significant address bit to the display buffer. Successive scan lines of the display image are displaced in 8KB of memory. This bit allows compatibility with the graphics modes of earlier adapters.

Line Compare Register

This register has an index of hex 18; its address is hex 03D5 or 03B5.

```
  7    6    5    4    3    2    1    0
┌───────────────────────────────────────┐
│             Line Compare              │
└───────────────────────────────────────┘
```

Figure 2-66. Line Compare Register, Index Hex 18

The Line Compare Register (bits 7 – 0) contains the 8 low-order bits of the 10-bit compare target. When the vertical counter reaches the target value, the internal start address of the line counter is cleared. This creates a split screen in which the lower screen is immune to scrolling. Bit 8 is in the Overflow register (index hex 07), and bit 9 is in the Maximum Scan Line register (index hex 09).

Graphics Controller Registers

The Address register for the graphics controller is at address hex 03CE. The data registers are at address hex 03CF. All registers are read/write.

Register Name	Address (Hex)	Index (Hex)
Address	03CE	
Set/Reset	03CF	00
Enable Set/Reset	03CF	01
Color Compare	03CF	02
Data Rotate	03CF	03
Read Map Select	03CF	04
Graphics Mode	03CF	05
Miscellaneous	03CF	06
Color Don't Care	03CF	07
Bit Mask	03CF	08

Figure 2-67. Graphics Controller Register Overview

Address Register

The Address register is at address hex 03CE. This register is loaded with the index value that points to the desired data register within the graphics controller.

7	6	5	4	3	2	1	0
–	–	–	–	Index			

– : Set to 0, Undefined on Read

Figure 2-68. Graphics Controller Address Register, Hex 03CE

The register field is defined as follows:

Index The Index field (bits 3 – 0) contains the index value that points to the data registers.

Set/Reset Register

This register has an index of hex 00; its address is hex 03CF.

7	6	5	4	3	2	1	0
–	–	–	–	SR3	SR2	SR1	SR0

–	Set to 0, Undefined on Read
SR3	Set/Reset Map 3
SR2	Set/Reset Map 2
SR1	Set/Reset Map 1
SR0	Set/Reset Map 0

Figure 2-69. Set/Reset Register, Index Hex 00

The register fields are defined as follows:

SR3, SR2, SR1, SR0
When write mode 0 is selected, the system writes the value of each set/reset field (bits 3, 2, 1, or 0) to its respective memory map. For each write operation, the set/reset bit, if enabled, is written to all 8 bits within that map. Set/reset operation can be enabled on a map-by-map basis through the Enable Set/Reset register.

Enable Set/Reset Register

The index for this register is hex 01; its address is hex 03CF.

7	6	5	4	3	2	1	0
–	–	–	–	ESR3	ESR2	ESR1	ESR0

–	:	Set to 0, Undefined on Read
ESR3	:	Enable Set/Reset Map 3
ESR2	:	Enable Set/Reset Map 2
ESR1	:	Enable Set/Reset Map 1
ESR0	:	Enable Set/Reset Map 0

Figure 2-70. Enable Set/Reset Register, Index Hex 01

The register fields are defined as follows:

ESR3, ESR2, ESR1, ESR0

These fields (bits 3, 2, 1, and 0) enable the set/reset function used when write mode 0 is selected in the Graphics Mode register (index hex 05). When set to 1, the respective memory map receives the value specified in the Set/Reset register. When Set/Reset is not enabled for a map, that map receives the value sent by the system.

Color Compare Register

This register has an index of hex 02; its address is hex 03CF.

7	6	5	4	3	2	1	0
–	–	–	–	CC3	CC2	CC1	CC0

–	:	Set to 0, Undefined on Read
CC3	:	Color Compare Map 3
CC2	:	Color Compare Map 2
CC1	:	Color Compare Map 1
CC0	:	Color Compare Map 0

Figure 2-71. Color Compare Register, Index Hex 02

The register fields are defined as follows:

CC3, CC2, CC1, CC0

These color compare map fields (bits 3, 2, 1, and 0) make up the 4-bit color value to be compared when the read mode bit in the Graphics Mode register is set to 1. When the system does a memory read, the data returned from the memory cycle will be a 1 in each bit position where the four maps equal the Color Compare register. If the read mode bit is 0, the data is returned without comparison.

The color compare bit is the value that all bits of the corresponding map's byte are compared with. Each of the 8 bit positions in the selected byte are compared across the four maps, and a 1 is returned in each position where the bits of all four maps equal their respective color compare values.

Data Rotate Register

This register has an index of hex 03; its address is hex 03CF.

7	6	5	4	3	2	1	0
–	–	–	FS		RC		

–	:	Set to 0, Undefined on Read
FS	:	Function Select
RC	:	Rotate Count

Figure 2-72. Data Rotate Register, Index Hex 03

The register fields are defined as follows:

FS Data written to the video buffer can be operated on logically by data already in the system latches. The Function Select field (bits 4, 3) determines whether and how this is done.

Data can be any of the choices selected by the write mode bits except system latches, which cannot be modified. If rotated data is selected also, the rotate is performed before the logical operation. The logical operations selected are shown in the following table.

FS Field (binary)	**Function**
0 0	Data Unmodified
0 1	Data ANDed with Latched Data
1 0	Data ORed with Latched Data
1 1	Data XORed with Latched Data

Figure 2-73. Operation Select Bit Definitions

RC In write mode 0, the Rotate Count field (bits 2–0) selects the number of positions the system data is rotated to the right during a system memory write operation. To write data that is not rotated in mode 0, all bits are set to 0.

Read Map Select Register

This register has an index of hex 04; its address is hex 03CF.

7	6	5	4	3	2	1	0
–	–	–	–	–	–	MS	

```
 – : Set to 0, Undefined on Read
MS : Map Select
```

Figure 2-74. Read Map Select Register, Index Hex 04

The register field is defined as follows:

MS The Map Select field (bits 1, 0) selects the memory map for system read operations. This register has no effect on the color compare read mode. In odd/even modes, the value can be a binary 00 or 01 to select the chained maps 0, 1 and the value can be a binary 10 or 11 to select the chained maps 2, 3.

Graphics Mode Register

This register has an index of hex 05; its address is hex 03CF.

7	6	5	4	3	2	1	0
–	C256	SR	OE	RM	–	WM	

–	: Set to 0, Undefined on Read
C256	: 256 - Color Mode
SR	: Shift Register Mode
OE	: Odd/Even
RM	: Read Mode
WM	: Write Mode

Figure 2-75. Graphics Mode Register, Index Hex 05

The register fields are defined as follows:

C256 When set to 0, the 256-Color Mode field (bit 6) allows bit 5 to control the loading of the shift registers. When set to 1, this field causes the shift registers to be loaded in a manner that supports the 256-color mode.

SR When set to 1, the Shift Register Mode field (bit 5) directs the shift registers in the graphics controller to format the serial data stream with even-numbered bits from both maps on even-numbered maps, and odd-numbered bits from both maps on the odd-numbered maps. This bit is used for modes 4 and 5.

OE When set to 1, the Odd/Even field (bit 4) selects the odd/even addressing mode used by the IBM Color/Graphics Monitor Adapter. Normally, the value here follows the value of Memory Mode register bit 2 in the sequencer.

RM When the Read Mode field (bit 3) is set to 1, the system reads the results of the comparison of the four memory maps and the Color Compare register.

When set to 0, the system reads data from the memory map selected by the Read Map Select register, or by the 2 low-order bits of the memory address (this selection depends on the chain-4 bit in the Memory Mode register of the sequencer).

WM The Write Mode field (bits 1, 0) determines the write mode selected. The write mode selected and its operation are defined in the following figure. The logic operation specified by the function select bits is performed on system data for modes 0, 2, and 3.

WM Field (binary)	**Mode Description**
0 0	Each memory map is written with the system data rotated by the count in the Data Rotate register. If the set/reset function is enabled for a specific map, that map receives the 8-bit value contained in the Set/Reset register.
0 1	Each memory map is written with the contents of the system latches. These latches are loaded by a system read operation.
1 0	Memory map *n* (0 through 3) is filled with 8 bits of the value of data bit *n*.
1 1	Each memory map is written with the 8-bit value contained in the Set/Reset register for that map (the Enable Set/Reset register has no effect). Rotated system data is ANDed with the Bit Mask register to form an 8-bit value that performs the same function as the Bit Mask register in write modes 0 and 2 (see also Bit Mask register on page 2-88).

Figure 2-76. Write Mode Definitions

Miscellaneous Register

This register has an index of hex 06; its address is hex 03CF.

7	6	5	4	3	2	1	0
-	-	-	-	MM		OE	GM

-	: Set to 0, Undefined on Read
MM	: Memory Map
OE	: Odd/Even
GM	: Graphics Mode

Figure 2-77. Miscellaneous Register, Index Hex 06

The register fields are defined as follows:

MM The Memory Map field (bits 3, 2) controls the mapping of the regenerative buffer into the system address space. The bit functions are defined in the following figure.

MM Field (binary)	**Addressing Assignment**
0 0	A0000 for 128KB
0 1	A0000 for 64KB
1 0	B0000 for 32KB
1 1	B8000 for 32KB

Figure 2-78. Video Memory Assignments

OE When set to 1, the Odd/Even field (bit 1) directs the system address bit, A0, to be replaced by a higher-order bit. The odd map is then selected when A0 is 1, and the even map when A0 is 0.

GM The Graphics Mode field (bit 0) controls alphanumeric mode addressing. When set to 1, this bit selects graphics modes, which also disables the character generator latches.

Color Don't Care Register

This register has an index of hex 07; its address is hex 03CF.

7	6	5	4	3	2	1	0
–	–	–	–	M3X	M2X	M1X	M0X

– :	Set to 0, Undefined on Read
M3X :	Map 3 is Don't Care
M2X :	Map 2 is Don't Care
M1X :	Map 1 is Don't Care
M0X :	Map 0 is Don't Care

Figure 2-79. Color Don't Care Register, Index Hex 07

The register fields are defined as follows:

M3X, M2X, M1X, M0X
These map don't care fields (bits 3, 2, 1, and 0) select whether a map is going to participate in the color compare cycle. When set to 1, the bits in that map are compared.

Bit Mask Register

This register has an index of hex 08; its address is hex 03CF.

```
  7   6   5   4   3   2   1   0
+-------------------------------+
|           Bit Mask            |
+-------------------------------+
```

Figure 2-80. Bit Mask Register, Index Hex 08

When a bit in the Bit Mask register (bits 7 – 0) is set to 1, the corresponding bit position in each map can be changed. When a bit is set to 0, the bit position in the map is masked to prevent change, provided that the location being written was the last location read by the system microprocessor.

The bit mask applies to write modes 0 and 2. To preserve bits using the bit mask, data must be latched internally by reading the location. When data is written to preserve the bits, the most current data in the latches is written in those positions. The bit mask applies to all maps simultaneously.

Attribute Controller Registers

Each register for the attribute controller has two addresses. Address hex 03C0 is the write address and hex 03C1 is the read address. The individual data registers are selected by writing their index to the Address register (hex 03C0).

Register Name	Write Address	Read Address	Index
Address	03C0	03C0	–
Internal Palette	03C0	03C1	00 – 0F
Attribute Mode Control			10
Overscan Color			11
Color Plane Enable			12
Horizontal PEL Panning			13
Color Select			14

Figure 2-81. Attribute Controller Register Addresses

Address Register

This read/write register is at address hex 03C0.

The attribute controller registers do not have an input bit to control selection of the address and data registers. An internal address flip-flop controls this selection. Reading Input Status Register 1 clears the flip-flop and selects the Address register.

After the Address register has been loaded with the index, the next write operation to 03C0 loads the data register. The flip-flop toggles for each write operation to address hex 03C0. It does not toggle for Read operations to 03C0 or 03C1. (Also see "VGA Programming Considerations" on page 2-96.)

7	6	5	4	3	2	1	0
–	–	IPAS	Index				

```
   – : Set to 0, Undefined on Read
IPAS : Internal Palette Address Source
```

Figure 2-82. Address Register, Hex 03C0

The register fields are defined as follows:

IPAS The Internal Palette Address Source field (bit 5) is set to 0 to load color values to the registers in the internal palette. It is set to 1 for normal operation of the attribute controller.

Note: Do not access the internal palette while this bit is set to 1. While this bit is 1, the Type 1 video subsystem disables accesses to the palette; however, the Type 2 does not, and the actual color value addressed cannot be ensured.

Index The Index field (bits 4 – 0) contains the index to the data registers in the attribute controller.

Internal Palette Registers 0 through F

These registers are at indexes hex 00 through 0F. Their write address is hex 03C0; their read address is hex 03C1.

7	6	5	4	3	2	1	0
–	–	P5	P4	P3	P2	P1	P0

```
        - : Set to 0, Undefined on Read
 P5 to P0 : Palette Data
```

Figure 2-83. Internal Palette Registers, Index Hex 00 - 0F

The register fields are defined as follows:

P5 – P0 These 6-bit registers (bits 5 – 0) allow a dynamic mapping between the text attribute or graphic color input value and the display color on the CRT screen. When set to 1, this bit selects the appropriate color. The Internal Palette registers should be modified only during the vertical retrace interval to avoid problems with the displayed image. These internal palette values are sent off-chip to the video DAC, where they serve as addresses into the DAC registers. (Also see the attribute controller block diagram on page 2-10.)

Note: These registers can be accessed only when bit 5 in the Address register is set to 0. When the bit is 1, writes are "don't care" and reads return undefined data.

Attribute Mode Control Register

This read/write register is at index hex 10. Its write address is hex 03C0; its read address is hex 03C1.

7	6	5	4	3	2	1	0
PS	PW	PP	–	EB	ELG	ME	G

–	: Set to 0, Undefined on Read
PS	: P5, P4 Select
PW	: PEL Width
PP	: PEL Panning Compatibility
EB	: Enable Blink/-Select Background Intensity
ELG	: Enable Line Graphics Character Code
ME	: Mono Emulation
G	: Graphics/-Alphanumeric Mode

Figure 2-84. Attribute Mode Control Register, Index Hex 10

The register fields are defined as follows:

PS The P5, P4 Select field (bit 7) selects the source for the P5 and P4 video bits that act as inputs to the video DAC. When set to 0, P5 and P4 are the outputs of the Internal Palette registers. When set to 1, P5 and P4 are bits 1 and 0 of the Color Select register. For more information, see "VGA Programming Considerations" on page 2-96.

PW When the PEL Width field (bit 6) is set to 1, the video data is sampled so that 8 bits are available to select a color in the 256-color mode (hex 13). This bit is set to 0 in all other modes.

PP When the PEL Panning Compatibility field (bit 5) is set to 1, a successful line-compare in the CRT controller forces the output of the PEL Panning register to 0 until a vertical synchronization occurs, at which time the output returns to its programmed value. This bit allows a selected portion of a screen to be panned.

When set to 0, line compare has no effect on the output of the PEL Panning register.

EB	When the Enable Blink/—Select Background Intensity field (bit 3) is set to 0, the most-significant bit of the attribute selects the background intensity (allows 16 colors for background). When set to 1, this bit enables blinking.
ELG	When the Enable Line Graphics Character Code field (bit 2) is set to 0, the ninth dot will be the same as the background. When set to 1, this bit enables the special line-graphics character codes for the monochrome emulation mode. This emulation mode forces the ninth dot of a line graphic character to be identical to the eighth dot of the character. The line-graphics character codes for the monochrome emulation mode are hex C0 through hex DF. For character fonts that do not use these line-graphics character codes, bit 2 should be set to 0 to prevent unwanted video information from displaying on the CRT screen. BIOS will set this bit, the correct dot clock, and other registers when the 9-dot alphanumeric mode is selected.
ME	When the Mono Emulation field (bit 1) is set to 1, monochrome emulation mode is selected. When set to 0, color emulation mode is selected.
G	When the Graphics/—Alphanumeric Mode field (bit 0) is set to 1, the graphics mode of operation is selected.

Overscan Color Register

This read/write register is at index hex 11. Its write address is hex 03C0; its read address is hex 03C1. This register determines the border (overscan) color.

7	6	5	4	3	2	1	0
Overscan Color							

Figure 2-85. Overscan Color Register, Index Hex 11

The Overscan Color register (bits 7 – 0) selects the border color used in the 80-column alphanumeric modes and in the graphics modes other than modes 4, 5, and D.

Color Plane Enable Register

This read/write register is at index hex 12. Its write address is hex 03C0; its read address is hex 03C1.

7	6	5	4	3	2	1	0
–	–	–	–	ECP			

```
  – : Set to 0, Undefined on Read
ECP : Enable Color Plane
```

Figure 2-86. Color Plane Enable Register, Index Hex 12

The register field is defined as follows:

ECP Setting a bit in the Enable Color Plane field (bits 3 – 0) to 1 enables the corresponding display-memory color plane.

Horizontal PEL Panning Register

This read/write register is at index hex 13. Its write address is hex 03C0; its read address is hex 03C1.

7	6	5	4	3	2	1	0
–	–	–	–	HPP			

```
    – : Set to 0, Undefined on Read
  HPP : Horizontal PEL Panning
```

Figure 2-87. Horizontal PEL Panning Register, Index Hex 13

The register field is defined as follows:

HPP The Horizontal PEL Panning field (bits 3–0) selects the number of PELs that the video data is shifted to the left. PEL panning is available in both alphanumeric (A/N) and graphics modes. The following figure shows the number of PELs shifted for each mode.

	Number of PELs Shifted to the Left		
Register Value	**Mode Hex 13**	**A/N Modes ***	**All Other Modes**
0	0	1	0
1	–	2	1
2	1	3	2
3	–	4	3
4	2	5	4
5	–	6	5
6	3	7	6
7	–	8	7
8	–	0	–
* Only mode 7 and the A/N modes with 400 scan lines.			

Figure 2-88. Image Shifting

Color Select Register

This read/write register is at index hex 14. Its write address is hex 03C0; its read address is hex 03C1.

7	6	5	4	3	2	1	0
–	–	–	–	SC7	SC6	SC5	SC4

–	:	Set to 0, Undefined on Read
SC7	:	S_color 7
SC6	:	S_color 6
SC5	:	S_color 5
SC4	:	S_color 4

Figure 2-89. Color Select Register, Index Hex 14

The register fields are defined as follows:

SC7, SC6 In modes other than mode hex 13, the S_color 7 and S_color 6 fields (bits 3, 2) are the 2 most-significant bits of the 8-bit digital color value to the video DAC. In mode hex 13, the 8-bit attribute is the digital color value to the video DAC. These bits are used to switch rapidly between sets of colors in the video DAC. (For more information, see "VGA Programming Considerations" on page 2-96.)

SC5, SC4 The S_color 5 and S_color 4 fields (bits 1, 0) can be used in place of the P4 and P5 bits from the Internal Palette registers to form the 8-bit digital color value to the video DAC. Selecting these bits is done in the Attribute Mode Control register (index hex 10). These bits are used to switch rapidly between color sets within the video DAC.

VGA Programming Considerations

The following are some programming considerations for the VGA:

- The following rules must be followed to guarantee the critical timings necessary to ensure the proper operation of the CRT controller:
  - The value in the Horizontal Total register must be at least hex 19.
  - The minimum positive pulse width of the 'horizontal synchronization' signal must be 4 character clock units.
  - The End Horizontal Retrace register must be programmed such that the 'horizontal synchronization' signal goes to 0 at least 1 character clock time before the 'horizontal display enable' signal goes active.
  - The End Vertical Blanking register must be set to a value at least one horizontal scan line greater than the line-compare value.
- When PEL panning compatibility is enabled in the Attribute Mode Control register, a successful line compare in the CRT controller forces the output of the Horizontal PEL Panning register to 0's until a vertical synchronization occurs. When the vertical synchronization occurs, the output returns to the programmed value. This allows the portion of the screen indicated by the Line Compare register to be operated on by the Horizontal PEL Panning register.
- A write to the Character Map Select register becomes valid on the next whole character line. This prevents deformed character images when changing character generators in the middle of a character scan line.
- For mode hex 13, the attribute controller is configured so that the 8-bit attribute in video memory becomes the 8-bit address (P0 through P7) into the video DAC. The user should not modify the contents of the Internal Palette registers when using this mode.

- The following is the sequence for accessing the attribute data registers:
    1. Disable interrupts.
    2. Reset the flip-flop for the Attribute Address register.
    3. Write the index.
    4. Access the data register.
    5. Enable interrupts.
- The Color Select register in the attribute controller section allows rapid switching between color sets in the video DAC. Bit 7 of the Attribute Mode Control register controls the number of bits in the Color Select register used to address the color information in the video DAC (either 2 or 4 bits are used). By changing the value in the Color Select register, an application can switch color sets in graphics and alphanumeric modes. (Mode hex 13 does not use this feature.)

  Note: For multiple color sets, the user must load the color values.
- An application that saves the video state must store the 4 bytes of information contained in the system microprocessor latches in the graphics controller subsection. These latches are loaded with 32 bits from video memory (8 bits per map) each time the system reads from video memory. The application must:
    1. Use write mode 1 to write the values in the latches to a location in video memory that is not part of the display buffer, such as the last location in the address range.
    2. Save the values of the latches by reading them back from video memory.

       Note: If memory addressing is in the chain-4 or odd/even mode, reconfigure the memory as four sequential maps prior to performing the sequence above.

BIOS provides support for completely saving and restoring the video state. Refer to the *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference* for more information.

- The Horizontal PEL Panning register allows programs to control the starting position of the display area on the screen. The display area can be shifted to the left up to eight PEL positions. In single-byte shift modes, to pan to more than eight PEL positions, the CRT controller start address is incremented and the Horizontal PEL Panning register is reset to 0.

  In multiple shift modes, the byte-panning bits (in the Preset Row Scan register) are used as extensions to the Horizontal PEL Panning register. This allows panning across the width of the video output. For example, in the 32-bit shift mode, the byte pan and PEL-panning bits provide panning up to 31 bits. To pan from position 31 to 32, the CRT controller start address is incremented and the panning bits, both PEL and byte, are reset to 0.

  Further panning can be accomplished by changing the start-address value in the CRT controller registers, Start Address High and Start Address Low. The sequence is:

  1. Use the Horizontal PEL Panning register to shift the maximum number of bits to the left.
  2. Increment the start address.
  3. Set the Horizontal PEL Panning register so that no bits are shifted.

     The screen is shifted one PEL to the left of the position it was in at the end of Step 1. Repeat Step 1 through Step 3 as often as necessary.

- When operating in a mode with 200 scan lines, and using a split-screen application that scrolls a second screen on top of the first screen, the Line Compare register (CRT Controller register hex 19) must contain an even value. This is a requirement of the double scanning logic in the CRT controller.

- If the value in the Cursor Start register (CRT Controller register hex 0A) is greater than that in the Cursor End register (CRT Controller register hex 0B), the cursor is not displayed.

- In 8-dot character modes, the underline attribute produces a solid line across adjacent characters. In 9-dot character modes, the underline across adjacent characters is dashed. In 9-dot modes with the line-graphics characters (C0 through DF character codes), the underline is solid.

Programming the Registers

Each of the video components has an address register and a number of data registers. The data registers have addresses common to all registers for that component. The individual registers are selected by a pointer (index) in their Address register. To write to a data register, the Address register is loaded with the index of the desired data register, then the data register is loaded by writing to the common I/O address.

The general registers do not share a common address; they each have their own I/O address.

See "Video DAC to System Interface" on page 2-103 for details on programming the video DAC.

For compatibility with the IBM Enhanced Graphics Adapter (EGA), the internal video subsystem palette is programmed the same as the EGA. Using BIOS to program the palette produces a color compatible to that produced by the EGA. Mode hex 13 (256 colors) is programmed so that the first 16 locations in the DAC produce compatible colors.

When BIOS is used to load the color palette for a color mode and a monochrome display is attached, the color palette is changed. The colors are summed to produce shades of gray that allow color applications to produce a readable screen.

Modifying the following bits must be done while the sequencer is held in a synchronous reset through its Reset register. The bits are:

- Bits 3 and 0 of the Clocking Mode register
- Bits 3 and 2 of the Miscellaneous Output register.

RAM Loadable Character Generator

The character generator is RAM loadable and can support characters up to 32 scan lines high. Three character fonts are stored in BIOS, and one is automatically loaded when an alphanumeric mode is selected. The Character Map Select register can be programmed to redefine the function of bit 3 of the attribute byte to be a character-font switch. This allows the user to select between any two character sets residing in map 2, and gives the user access to 512 characters instead of 256. Character fonts can be loaded off line, and up to eight fonts can be loaded at any one time.

The structure of the character fonts is described in the following figure. The character generator is in map 2 and must be protected using the map mask function.

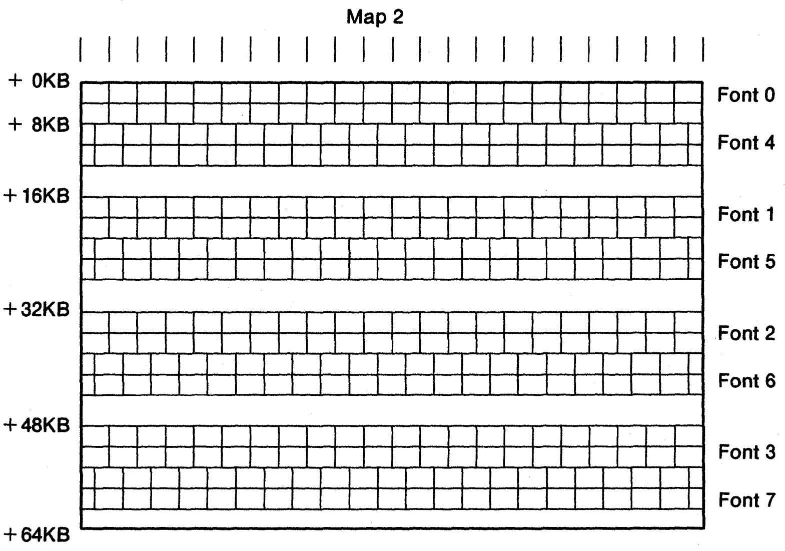


Figure 2-90. Character Table Structure

The following figure illustrates the structure of each character pattern. If the CRT controller is programmed to generate 16 row scans, then 16 bytes must be filled in for each character in the font. The example below assumes eight row scans per character.

Address	Byte Image								Data
CC * 32 + 0				X	X				18H
1			X	X	X	X			3CH
2		X	X			X	X		66H
3		X	X			X	X		66H
4		X	X	X	X	X	X		7EH
5		X	X			X	X		66H
6		X	X			X	X		66H
7		X	X			X	X		66H

*CC equals the value of the character code. For example, hex 41 equals and ASCII ″A″.

Figure 2-91. Character Pattern Example

Creating a Split Screen

The VGA hardware supports a split screen. The top portion of the screen is designated as screen A, and the bottom portion is designated as screen B, as in the following figure.

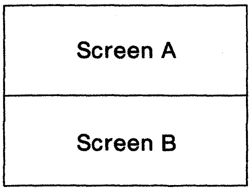


Figure 2-92. Split Screen Definition

The following figure shows the screen mapping for a system containing a 32KB alphanumeric storage buffer, such as the VGA. Information displayed on screen A is defined by the Start Address High and Low registers of the CRT controller. Information displayed on screen B always begins at video address hex 0000.

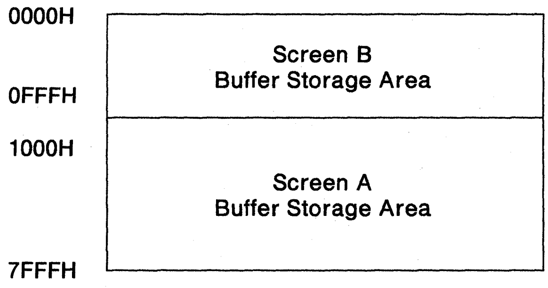


Figure 2-93. Screen Mapping within the Display Buffer Address Space

The Line Compare register of the CRT controller performs the split screen function. The CRT controller has an internal horizontal scan line counter and logic that compares the counter value to the value in the Line Compare register and clears the memory address generator when a comparison occurs. The linear address generator then sequentially addresses the display buffer starting at location 0. Each subsequent row address is determined by the 16-bit addition of the start-of-line latch and the Offset register.

Screen B can be smoothly scrolled onto the display by updating the Line Compare register in synchronization with the 'vertical retrace' signal. Screen B information is not affected by scrolling operations that use the Start Address registers to scroll through screen A information.

When PEL-panning compatibility is enabled (Attribute Mode Control register), a successful line comparison forces the output of the Horizontal PEL Panning register to 0's until vertical synchronization occurs. This feature allows the information on screen B to remain unaffected by PEL-panning operations on screen A.

Video Digital-to-Analog Converter

The video digital-to-analog converter (DAC) integrates the function of a color palette with three internal DACs for driving an analog display.

The DAC has 256 registers containing 18 bits each to allow the display of up to 256 colors from a possible 256k colors. Each output signal is driven by a 6-bit DAC.

Register Name	Read/ Write	Address (in Hex)
Palette Address (Write Mode)	R/W	03C8
Palette Address (Read Mode)	W	03C7
DAC State	R	03C7
Palette Data	R/W	03C9
PEL Mask	R	03C6

Figure 2-94. Video DAC Register

Device Operation

The palette address (P7 through P0) and the blanking input are sampled on the rising edge of the PEL clock. After three more PEL clock cycles, the video reflects the state of these inputs.

During normal operation, the palette address is used as a pointer to one of the 256 data registers in the palette. The value in each data register is converted to an analog signal for each of the three outputs (red, green, blue). The blanking input is used to force the video output to 0 volts. The blanking operation is independent of the palette operation.

Each data register is 18 bits wide: 6 bits each for red, green, and blue. The data registers are accessible through the system interface.

Video DAC to System Interface

The Palette Address register holds an 8-bit value that is used to address a location within the video DAC. The Palette Address register responds to two addresses; the address depends on the type of palette access, read or write. Once the address is loaded, successive accesses to the data register automatically increment the address register.

For palette write operations, the address for the Palette Address register is hex 03C8. A write cycle consists of writing three successive bytes to the Data register at address hex 03C9. The 6 least-significant bits of each byte are concatenated to form the 18-bit palette data. The order is red value first, then green, then blue.

For palette read operations, the address for the Palette Address register is hex 03C7 (in the read mode, the Palette Address register is write only). A read cycle consists of reading three successive bytes from the Data register at address hex 03C9. The 6 least-significant bits of each byte contain the corresponding color value. The order is red value first, then green, then blue.

If the Palette Address register is written to during a read or write cycle, a new cycle is initialized and the unfinished cycle is terminated. The effects of writing to the Data register during a read cycle or reading from the Data register during a write cycle are undefined and can change the palette contents.

The DAC State register is a read-only register at address hex 03C7. Bits 1 and 0 return the last active operation to the DAC. If the last operation was a read operation, both bits are set to 1. If the last operation was a write, both bits are set to 0.

Reading the Read Palette Address register at hex 03C8 or the DAC State register at hex 03C7 does not interfere with read or write cycles.

Programming Considerations

- As explained in "Video DAC to System Interface" on page 2-103, the effects of writing to the Data register during a read cycle or reading from the Data register during a write cycle are undefined and can change the palette contents. Therefore, the following sequence must be followed to ensure the integrity of the color palette during accesses to it:

  1. Disable interrupts.
  2. Write the address to PEL Address register.
  3. Write or read three bytes of data.
  4. Go to Step 2, repeat for the desired number of locations.
  5. Enable interrupts.

  Note: All accesses to the DAC registers are byte-wide I/O operations.

- To prevent "snow" on the screen, an application reading data from or writing data to the DAC registers should ensure that the blank input to the DAC is asserted. This can be accomplished either by restricting data transfers to retrace intervals (use Input Status Register 1 to determine when retrace is occurring) or by using the screen off bit located in the Clocking Mode register in the sequencer.

  Note: BIOS provides read and write interfaces to the video DAC.

- Do not write to the PEL Mask register (hex 03C6). Palette information can be changed as a result. This register is correctly initialized to hex FF during a mode set.

VGA Video Extensions

The video extensions provide a means of transferring video information between the base video subsystem and an auxiliary video adapter.

The video extensions consist of:

- The auxiliary video extension
- The base video extension
- The auxiliary video signals.

The base video is provided by the video subsystem integrated onto the system board, or, when not provided on the system board, by a suitable video adapter. Such an adapter can provide a Micro Channel connector with the base video extension. Video adapters supporting the base video extension must provide the VGA function as the default. For detailed connector dimensions, see Micro Channel Adapter Design in *Personal System 2 Hardware Interface Technical Reference - Architectures*.

The buffers for the base video can be turned off to allow video output from the auxiliary video to be sent through the base video DAC to the display. The video extension can be driven in only one direction at a time.

Note: The video extension supports only the VGA function (see Figure 2-97 on page 2-110).

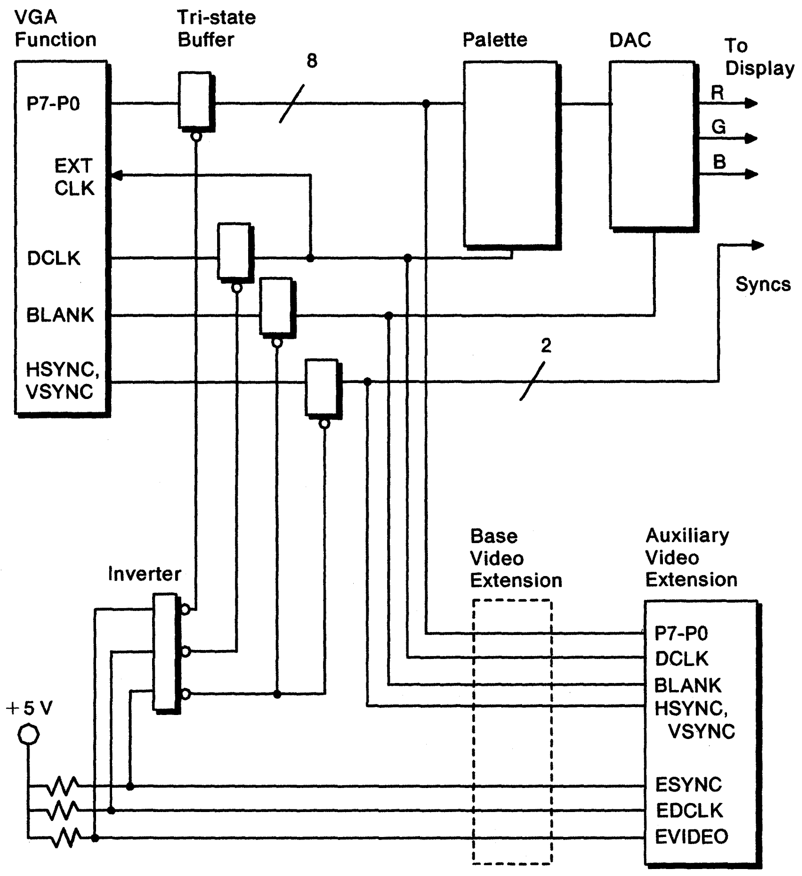


Figure 2-95. Auxiliary Video Connector Interface

Auxiliary Video Extension

This extension provides a video adapter with the ability to access the resources of the base video subsystem.

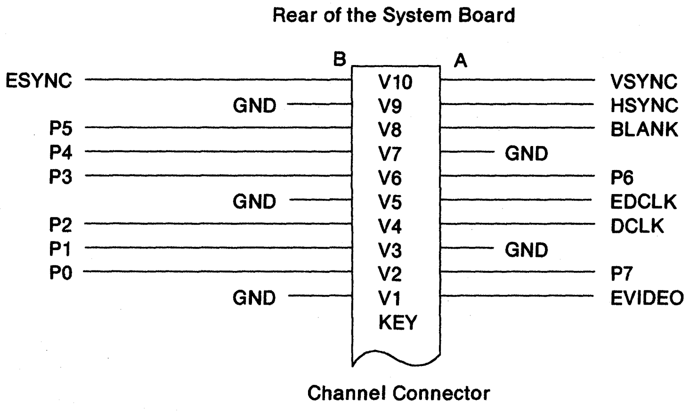


Figure 2-96. Video Extension

The auxiliary video extension is an optional part of the 16- or 32-bit Micro Channel connector.

Note: For more information on the auxiliary and base video connectors and extensions, see Micro Channel Architectures in *Personal System 2 Hardware Interface Technical Reference - Architectures*.

Base Video Extension

This extension is for adapters that provide the base video subsystem. Only systems without a base video subsystem on the system board have a connector with this extension. The base video extension signals and auxiliary video extension signals are identical.

Video adapters supporting the base video extension must provide the VGA function as the default.

The base video extension is an optional part of the 16- or 32-bit Micro Channel connector and is positioned at the end of the matched memory extension.

Video Extension Signal Descriptions

The following are signal descriptions for the auxiliary and base video extensions of the channel connector.

VSYNC: Vertical Synchronization: This signal is the vertical synchronization signal to the display. Also see the ESYNC description.

HSYNC: Horizontal Synchronization: This signal is the horizontal synchronization signal to the display. Also see the ESYNC description.

BLANK: Blanking Signal: This signal is connected to the BLANK input of the video DAC. When active (0 V dc), this signal tells the DAC to drive its analog color outputs to 0 V dc. Also see the ESYNC description.

P7 – P0: Palette Bits: These eight signals contain video information and comprise the PEL address inputs to the video DAC. See also the EVIDEO description.

DCLK: Dot Clock: This signal is the PEL clock used by the DAC to latch the digital video signals, P7 through P0. The signals are latched into the DAC on the rising edge of DCLK.

This signal is driven through the EXTCLK input to the VGA when DCLK is driven by the adapter. If an adapter is providing the clock, it must also provide the video data to the DAC. Also see the EDCLK description.

ESYNC: External Synchronization: This signal is the output-enable signal for the buffer that drives BLANK, VSYNC, and HSYNC. ESYNC is tied to +5 V dc through a pull-up resistor. When ESYNC is high, the VGA drives BLANK, VSYNC, and HSYNC. When ESYNC is pulled low, the adapter drives BLANK, VSYNC, and HSYNC.

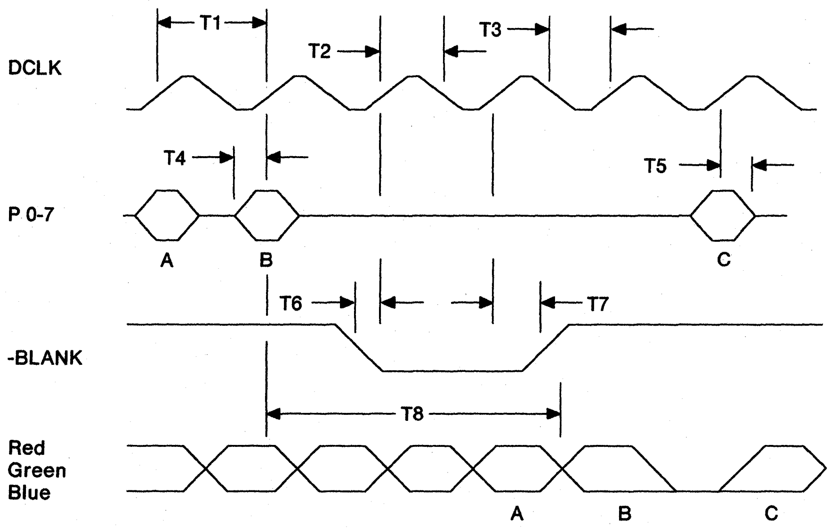
EVIDEO: External Video: This signal is the output-enable signal for the buffer that drives P7 through P0. EVIDEO is tied to +5 V dc through a pull-up resistor. When EVIDEO is high, the VGA drives P7 through P0. When it is pulled low, the adapter drives P7 through P0.

EDCLK: External Dot Clock: This signal is the output-enable signal for the buffer that drives DCLK. EDCLK is tied to +5 V dc through a pull-up resistor.

When EDCLK is high, the VGA is the source of DCLK to the DAC and the adapter. The Miscellaneous Output register should not select clock source 2 (010 binary) when EDCLK is high.

When EDCLK is pulled low, the adapter drives DCLK. If the adapter is driving the clock, it must also provide the video data to the DAC, and the Miscellaneous Output register must select clock source 2 (010 binary).

Video Extension Signal Timing



Symbol	Description	Minimum (ns)	Maximum (ns)
T1	PEL Clock Period	28	10,000
T2	Clock Pulse Width High	7	10,000
T3	Clock Pulse Width Low	9	10,000
T4	PEL Set-Up Time	4	–
T5	PEL Hold Time	4	–
T6	Blank Set-Up Time	4	–
T7	Blank Hold Time	4	–
T8	Analog Output Delay	3(T1) + 5	3(T1) + 30

Figure 2-97. Video Extension Signal Timing (DAC Signals)

Section 3. XGA Function

XGA Function Introduction

The XGA function is generated by the type 2 video subsystem. This chapter describes the capabilities and operation of the XGA function.

The XGA video subsystem has three modes:

- VGA
- 132-column text
- Extended Graphics.

VGA Mode

In VGA mode, the XGA video subsystem is VGA register compatible, as defined in the VGA function description.

132-Column Text Mode

In this mode, text is displayed in 132 vertical columns using 200, 350, or 400 scan lines. Each character is eight PELs wide.

Extended Graphics Mode

Extended Graphics mode provides the following software and hardware support.

IBM PS/2 8514/A Adapter Interface Compatibility

Compatibility is provided through the XGA Adapter Interface, a device driver supplied with the subsystem as programming support for applications operating in the disk operating system (DOS) environment.

High Resolution Support

Depending on the display attached and the size of video memory installed, the image on a screen can be defined using 1024 PELs and 768 scan lines with 256 colors.

Direct Color Mode

In this mode, each 16-bit PEL in video memory specifies the color of the PEL directly. This allows 65,536 colors to be displayed using 640 PELs and 480 scan lines.

Packed PEL Format

In the packed PEL format, reads and writes to the video memory access all the data that defines a PEL (or PELs) in a single operation.

Hardware Sprite

The sprite is a 64 x 64 PEL image. When enabled, it overlays the picture that is being displayed. It can be positioned anywhere on the display without affecting the contents of video memory.

Display Identification

Signals from the attached display identify its characteristics. Applications use this information to determine the maximum resolution and whether the display is color or monochrome.

Coprocessor

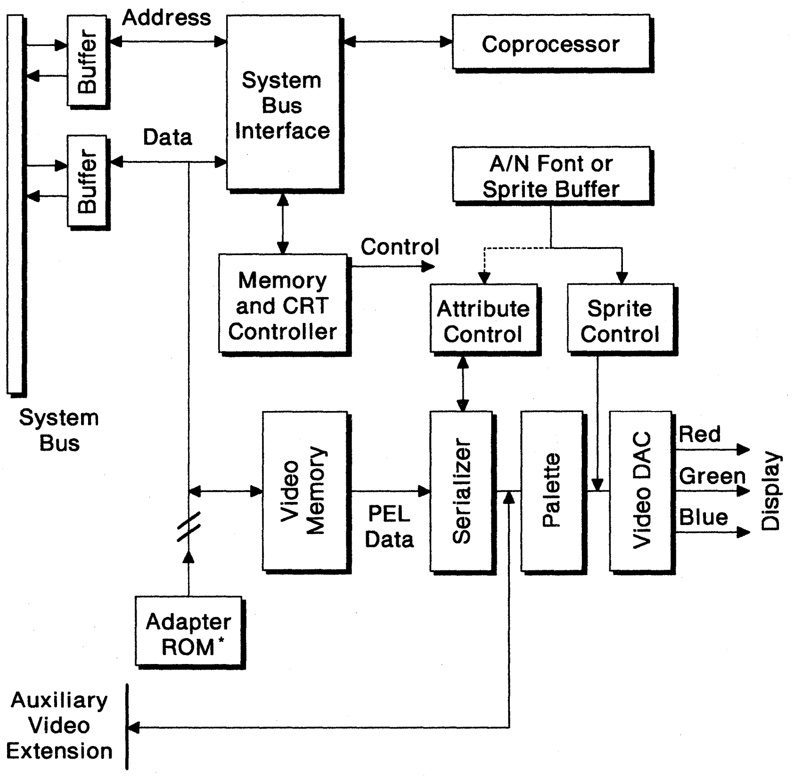
The coprocessor provides hardware drawing-assist functions throughout real or virtual memory. The following functions can be used with the XGA Adapter Interface:

- PEL-block and bit-block transfers (PxBlt)
- Line drawing
- Area filling
- Logical and arithmetic mixing
- Map masking
- Scissoring
- X and Y axis addressing.

XGA Components

The XGA video subsystem components include:

- System bus interface
- Memory and CRT controller
- Coprocessor
- Video memory
- Attribute controller
- Sprite controller
- Alphanumeric (A/N) font and sprite buffer
- Serializer
- Palette
- Video digital-to-analog convertor (DAC).



* ROM is only present on adapters.

Figure 3-1. XGA Video Subsystem

System Bus Interface

The system bus interface controls the interface between the video subsystem and the system microprocessor. It decodes the addresses for VGA and XGA I/O registers, the memory addresses for the coprocessor memory-mapped registers, and video memory.

It also provides the busmaster function, and determines whether the system data bus is 16 or 32 bits wide.

Memory and CRT Controller

The memory and CRT controller controls access to video memory by the system microprocessor, displays the contents of video memory on the display, and provides support for the VGA and 132-column text modes.

Coprocessor

The coprocessor provides hardware drawing-assist functions. These functions can be performed on graphics data in video memory and system memory.

The coprocessor updates the video memory independently of the system microprocessor. Instructions are written to a set of memory-mapped registers; the coprocessor then executes the drawing function.

The coprocessor functions are:

PEL-Block or Bit-Block Transfers
Transfers a bit map, or part of a bit map, from one location to another:

- Within video memory
- Within system memory
- Between system and video memory.

Line Drawing
Draws lines, with a programmable style, into a bit map in video memory or system memory.

Area Fill
Fills an outlined area in video memory or system memory with a programmable pattern.

Logical and Arithmetic Mixing
Provides logical and arithmetic operations for use with data in video memory or system memory.

Map Masking
Controls updates to each PEL for all drawing functions.

Scissoring
Provides a rectangular-mask function for use instead of the mask map.

X and Y Axis Addressing
Allows a PEL to be specified by its X and Y coordinates within a PEL map, instead of by its linear address in memory.

Video Memory

The video subsystem uses a dual-port video memory to store on-screen data, so that video memory can be read serially to display its contents as the data is being updated.

Attribute Controller

The attribute controller works with the memory and CRT controller to control the color selection and character generation in the 132-column text mode and VGA text modes.

Sprite Controller

The sprite controller is used to display and control the position and image of the sprite (cursor). The sprite is not available in 132-column text mode or VGA modes.

The Serializer, Palette, and Video DAC

The serializer takes data from the serial port of video memory in 16- or 32-bit widths (depending on the size of video memory) and converts it to a serial stream of PEL data. The PEL data addresses a palette location, which contains the color value. The color value is passed to the DAC, which converts the digital information into red, green, and blue analog signals for the display.

Alphanumeric (A/N) Font and Sprite Buffer

This buffer holds the character fonts in 132-column text mode and VGA modes. It also stores the sprite image in Extended Graphics mode.

Modes Of Operation

The 132-column text mode and all VGA modes are available on the XGA video subsystem.

In Extended Graphics mode, the size of the video memory determines the screen resolutions and number of colors that are supported as shown below:

Video Memory Installed	Resolution (PELs)	Colors (maximum)
512KB	640 x 480	256
	1024 x 768	16
1MB	640 x 480	65,536
	1024 x 768	256

Figure 3-2. XGA Color Resolution

Compatibility

8514/A Adapter Interface

The XGA function is *not* hardware register compatible with the 8514/A Adapter Interface. Applications written directly to the register-level interface of the 8514/A Adapter Interface do not run.

The XGA function is 8514/A Adapter Interface compatible in the DOS environment through a DOS Adapter Interface driver supplied with the XGA video subsystem.

Applications written to the 8514/A DOS Adapter Interface should run unchanged with the XGA Adapter Interface. The following differences, however, should be noted:

OS/2 protect mode adapter interface
An XGA Adapter Interface driver is not available for the OS/2 protect mode.

640 x 480, 4 + 4 mode with 512KB display buffer
This is not an Extended Graphics mode, but applications using this mode and written to the rules for the 8514/A Adapter Interface will run.

Dual-display buffer applications
8514/A applications using VGA or other advanced function modes that rely on two separate video display buffers do not run on a single-display configuration. These applications run correctly with two video subsystems (when one is an XGA), and each has a display attached.

Nondisplay memory
The XGA and 8514/A nondisplay (off-screen) memory are mapped differently. Applications using areas of the off-screen memory for storage may not run.

Adapter interface code size
The XGA Adapter Interface code size is larger than that for the 8514/A. This reduces the amount of system memory available to applications.

Adapter interface enhancements
The XGA Adapter Interface is a superset of that provided with the 8514/A. Any 8514/A applications using invalid specifications of parameter blocks may trigger some of the additional functions provided by the XGA Adapter Interface.

Use of LIM EMS drivers
Applications written to the 8514/A Adapter Interface that locate resources, such as bit maps or font definitions, in LIM EMS memory, and pass addresses of these resources to the adapter interface, require a LIM driver that has implemented the Physical Address Services Interface for busmasters.

Time-dependent applications
Some XGA and 8514/A functions run at different speeds. Applications that rely on a fixed performance may be affected by these differences.

XGA Adapter Interface directory and module name
The directory and module name of the XGA Adapter Interface \XGAPCDOS\XGAAIDOS.SYS is different from that of the 8514/A \HDIPCDOS\HDILOAD.EXE. Applications written to rely on the existence of either the specific 8514/A module name or directory do not run on the XGA Adapter Interface.

8514/A and XGA Adapter Interface code type

The XGA Adapter Interface is implemented as a .SYS device driver. The 8514/A Adapter Interface is implemented as a terminate and stay resident program. Applications written to rely on the adapter interface as a terminate and reside program do not run on the XGA Adapter Interface.

LIM EMS Drivers

The XGA coprocessor memory-mapped registers are located in system memory address space. They reside in the top 1KB of an 8KB block of memory assigned to the XGA subsystem. The lower 7KB of this block is used to address the ROM of an XGA subsystem on an adapter card.

Although an XGA subsystem integrated on the system board does not have a subsystem ROM, an 8KB block of memory is allocated to it to support the coprocessor memory-mapped registers. While the lower 7KB of this 8KB block does not contain any memory, the memory-mapped registers are accessed in the top 1KB of the block.

Applications or drivers, such as LIM EMS drivers that scan memory addresses looking for RAM or ROM signatures, may assume incorrectly that all 8KB of memory is available for use.

The location of the 8KB block of memory assigned to the XGA subsystem can be determined using the System Unit Reference diskette. See the LIM driver installation instructions for details on how to avoid address conflicts.

VGA compatibility

The XGA subsystem is register compatible with the VGA, as defined in the VGA function description (see "Setting the XGA Subsystem Mode" on page 3-182 for switching between the different XGA subsystem modes).

132-Column Text Mode

In this mode the XGA subsystem is capable of displaying 132 8-PEL wide alphanumeric characters on the display. It is register compatible with the VGA, except for the following VGA CRT controller registers:

Horizontal Total
VGA requires that this register holds a value that is five less than the number of characters on a scan line. In 132-column text mode, this register requires a value that is one less than the number of characters on a scan line.

The End Horizontal Retrace
In 132-column text mode the End Horizontal Retrace field has no effect. Instead, Extended Graphics mode Horizontal Sync Pulse End register (index hex 1A) is used to give a larger horizontal count.

The Horizontal Retrace Delay field has no effect. Instead, Extended Graphics mode Horizontal Sync Pulse Position registers (index hex 1C and 1E) are used.

The End Horizontal Blanking, Bit 5 field continues to be effective.

See "Setting the XGA Subsystem Mode" on page 3-182 for starting 132-column text mode.

Extended Graphics Mode

Extended Graphics mode provides applications with high-resolution, a wide range of colors, and high-performance. The XGA coprocessor provides hardware assistance in drawing and moving data in video memory and in system memory. Extended Graphics mode is controlled using a bank of 16 I/O registers, and the coprocessor is controlled by a bank of 128 memory-mapped registers.

See "Locating the XGA Subsystem" on page 3-171 to locate the subsystem in I/O and memory space.

Display Controller

Video Memory Format

The XGA video memory appears to the system as a byte-addressable, packed array of PELs. The PELs may be 1, 2, 4, 8, or 16 bits long. The first PEL in memory is displayed at the top left corner of the screen. The next PEL is immediately to its right and so on. Addressing is not necessarily contiguous, going from one horizontal line to the next. Addressing depends on the values in the Display PEL Map Width registers. See "CRT Controller" on page 3-19.

Two orders of PELs are supported: Intel and Motorola.

To allow the XGA subsystem to function in either environment, the Memory Access Mode register (for display controller accesses) and the PEL Map n Format register (for coprocessor accesses) are used to make the PELs appear in the required order.

The two formats are described in the following paragraphs.

Intel Order: This table represents the first 3 bytes of the memory map in Intel order and shows the layout of the PELs within those bytes for all PEL sizes (bpp = bits-per-PEL).

PEL	**Byte = n + 2**								**Byte = n + 1**								**Byte = n + 0**							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Size = 1bpp																								
Number	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit significance	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Size = 2bpp																								
Number	11	11	10	10	9	9	8	8	7	7	6	6	5	5	4	4	3	3	2	2	1	1	0	0
Bit significance	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Size = 4bpp																								
Number	5	5	5	5	4	4	4	4	3	3	3	3	2	2	2	2	1	1	1	1	0	0	0	0
Bit significance	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
Size = 8bpp																								
Number	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Bit significance	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Size = 16bpp																								
Number	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit significance	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 3-3. Intel Order of the XGA Memory Map

Motorola Order: This table represents the first 3 bytes of the memory map in Motorola order and shows the layout of the PELs within those bytes for all PEL sizes (bpp = bits-per-PEL).

PEL	Byte = n + 0								Byte = n + 1								Byte = n + 2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Size = 1bpp																								
Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Bit significance	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Size = 2bpp																								
Number	0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	9	9	10	10	11	11
Bit significance	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Size = 4bpp																								
Number	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5
Bit significance	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
Size = 8bpp																								
Number	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
Bit significance	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Size = 16bpp																								
Number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Bit significance	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8

Figure 3-4. Motorola Order of the XGA Memory Map

PEL Color Mapping

In 1, 2, 4, or 8 bits-per-PEL modes, the palette address is the numerical value of the PEL.

In 16 bits-per-PEL mode (direct color), the color mapping is 5 bits red, 6 bits green, and 5 bits blue. See "Direct Color Mode" on page 3-27.

Border Color Mapping

In the border area of the display, the palette is addressed by the Border Color register (index hex 55). The border area is defined in "CRT Controller" on page 3-19.

Direct Access to Video Memory

An application can use normal memory accesses to read or write PELs in video memory. All bits of one or more PELs can be accessed in a single memory cycle.

System Apertures into Video Memory: The XGA subsystem video memory is accessed in system memory address space through three possible apertures:

4MB Aperture
This allows up to 4MB of video memory to be addressed consecutively. If an access is made at an offset higher than the size of memory installed, no memory is written and undefined values are returned when read.

1MB Aperture
This allows up to 1MB of video memory to be addressed consecutively. If an access is made at an offset higher than the size of memory installed, no memory is written and undefined values are returned when read.

Note: To use the 1MB aperture, the Aperture Index register must be set to 0.

64KB Aperture
This allows up to 64KB of video memory to be addressed consecutively.

The aperture can be located at any 64KB section of the video memory using the Aperture Index register.

See "System Video Memory Apertures" on page 3-187 for details on locating and using these apertures.

CRT Controller

The CRT controller generates all timing signals required to drive the serializer and the display. It consists of two counters, one for horizontal parameters and one for vertical parameters, and a series of registers. The counters run continuously, and when the count-value reaches that specified in one of the associated registers, the event controlled by that register occurs.

See "Setting the XGA Subsystem Mode" on page 3-182 for mode tables, including CRT controller register values.

CRT Controller Register Interpretations

A representation of the function of each of the CRT controller registers is given in the following figure.

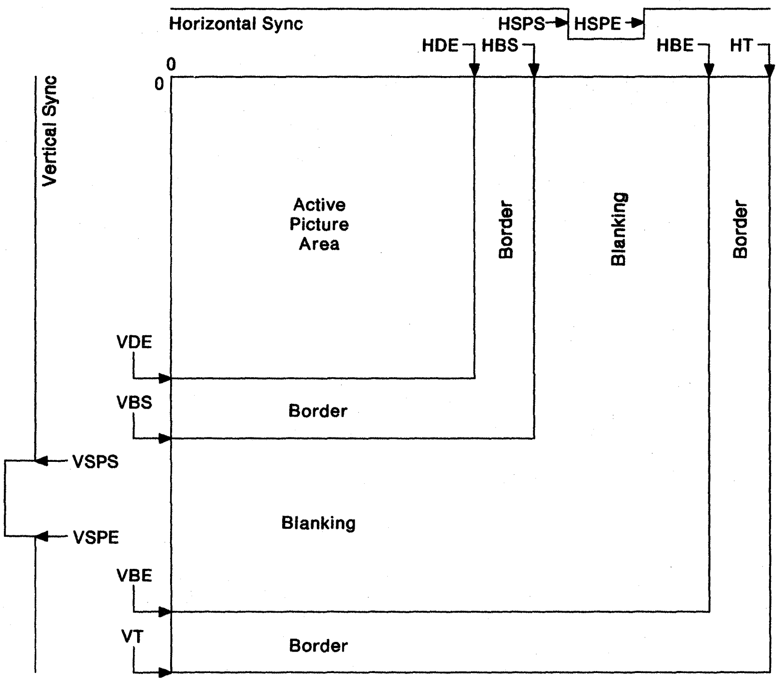


Figure 3-5. CRT Controller Register Definitions

The registers that control a horizontal scan of the display are:

HT	Horizontal Total register
HDE	Horizontal Display End register
HBS	Horizontal Blanking Start register
HBE	Horizontal Blanking End register
HSPS	Horizontal Sync Pulse Start register
HSPE	Horizontal Sync Pulse End register

The registers that control a vertical scan of the display are:

VT	Vertical Total register
VDE	Vertical Display End register
VBS	Vertical Blanking Start register
VBE	Vertical Blanking End register
VSPS	Vertical Sync Pulse Start register
VSPE	Vertical Sync Pulse End register

By using a system interrupt, the XGA subsystem can be programmed to inform the system microprocessor of the start and the end of the active picture area. An enable bit exists for each interrupt in the "Interrupt Enable Register (Address 21x4)" on page 3-34, and the "Interrupt Status Register (Address 21x5)" on page 3-36 contains a status bit for each interrupt.

Scrolling

Some or all of the displayed picture can be made to scroll. The first PEL displayed on the screen is controlled by the Display PEL Map Offset registers. These can be altered to a granularity of 8 bytes, giving coarse horizontal scrolling. Vertical scrolling is achieved by altering the Display PEL Map Offset registers in units of one line length. The line length is stored in the Display PEL Map Width registers. The value stored in the width registers is the amount of memory allocated to each line, not necessarily the physical length of the line being displayed.

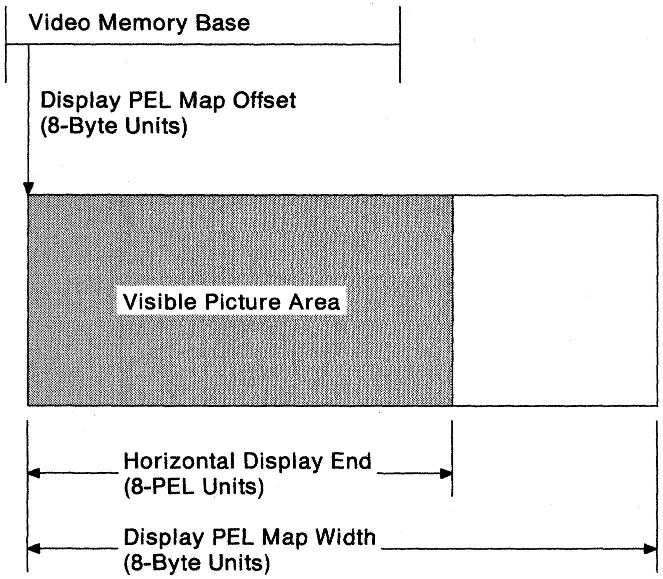

Video Memory Base
Display PEL Map Offset (8-Byte Units)
Visible Picture Area
Horizontal Display End (8-PEL Units)
Display PEL Map Width (8-Byte Units)

Figure 3-6. Display PEL Map Offset and Width Definitions

The Display PEL Map Width registers must be loaded with a value greater than or equal to the length of line being displayed. The most efficient use of video memory is achieved when the width value is made equal to the length of the line being displayed. However, it is often more convenient to load a width value that specifies the start of each line on a suitable address boundary.

An area at the bottom of the display can be prevented from scrolling by using the Vertical Line Compare registers (index hex 2C and 2D).

Sprite

The sprite is a 64 x 64-PEL image stored in the XGA subsystem alpha/sprite buffer. When active, the sprite overlays the picture that is displayed. Each PEL in the sprite can take on four values that can be used to achieve the effect of a colored marker of arbitrary shape.

Sprite Color Mapping

The sprite is stored as 2-bit packed PELs, using Intel format, in the sprite buffer. Address zero is at the top left corner of the sprite.

These 2-bit PELs determine the sprite appearance as shown in the following figure:

Bits **1 0**	**Sprite Effect**
0 0	Sprite color 0
0 1	Sprite color 1
1 0	Transparent
1 1	Complement

Figure 3-7. Sprite Appearance Defined by 2-bit PEL

The sprite effect definitions are as follows:

Sprite Colors 0 and 1
These colors are set by writing to the Sprite Color registers (index hex 38 through 3D).

Transparent
The underlying PEL color is displayed.

Complement
The ones complement of the underlying PEL color is displayed.

Sprite Buffer Accesses

The sprite buffer is written to by loading a number into the Sprite Index High and Sprite/Palette Index Low registers. These registers indicate the location of the first group of four sprite PELs to be updated (2 bits-per-PEL implies 4 PELs-per-byte). Then the first four PELs are written to the Sprite Data register. This stores the sprite PELs in the sprite buffer and automatically increments the index registers. Subsequent writes to the Sprite Data register load the remaining sprite PELs, four at a time.

The prefetch function is used to read from the sprite buffer. The index or address of the first sprite buffer location to be read is loaded into the index registers. Writing to either the Sprite Prefetch Index High or the Sprite/Palette Prefetch Index Low registers increments both registers as a single value. The first byte of the index must be written to a non-prefetch index register, and the second byte to the other prefetch index register. For example, write to Sprite Index High, then Sprite/Palette Prefetch Index Low.

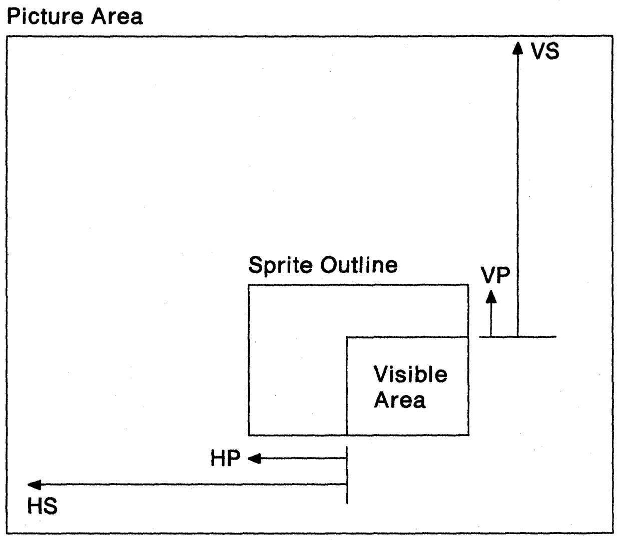
Writing to a prefetch index register loads the sprite data that is stored at the location specified in the index registers into a holding register, then increments the index registers as a single value. Reading the Sprite Data register returns the four sprite PELs that were prefetched, and loads the next four sprite PELs into the holding register. Subsequent reads from the Sprite Data register return the remaining sprite PELs, four at a time.

The sprite and the palette use the same hardware registers during reading and writing, so any task that is updating either the sprite or palette when an interrupt occurs must save and restore the following registers:

- Sprite/Palette Index Low register (index hex 60)
- Sprite Index High register (index hex 61)
- Palette Sequence register (index hex 66)
- Palette Red Prefetch register (index hex 67)
- Palette Green Prefetch register (index hex 68)
- Palette Blue Prefetch register (index hex 69)
- Sprite Prefetch register (index hex 6B).

Note: The Sprite/Palette Prefetch Index Low register (index hex 62) and Sprite Prefetch Index High register (index hex 63) must not be saved and restored.

Sprite Positioning



HS - Horizontal Sprite Start
HP - Horizontal Sprite Preset
VS - Vertical Sprite Start
VP - Vertical Sprite Preset

Figure 3-8. Sprite Positioning

The sprite position is controlled by Start and Preset registers. The Start registers control where the first displayed sprite PEL appears on the screen, and the Preset registers control which sprite PEL is first displayed within the 64 x 64 sprite definition. Using these registers, the sprite can be made to appear at any point in the picture area. If the sprite overlaps any edge, the part of the sprite outside the picture area is not visible (does not wrap). See "Sprite Handling" on page 3-220.

The XGA subsystem can be programmed to inform the system microprocessor when the last line of the sprite has been displayed on each frame using a sprite-display-complete system interrupt. An enable bit exists for each interrupt in the Interrupt Enable register, and the Interrupt Status register contains a status bit for each interrupt. See "Interrupt Enable Register (Address 21x4)" on page 3-34 and "Interrupt Status Register (Address 21x5)" on page 3-36 for the location of the bits.

Palette

The palette has 256 locations and each location contains three fields, one each for red, green, and blue. The palette is used to translate the PEL value into a displayed color.

Before the PEL value is used to address the palette, it is masked by the Palette Mask register. All bits in the PEL corresponding to 0's in the Palette Mask register are forced to 0 before reaching the palette.

Palette Accesses

The Palette Data register is 1 byte wide. Because each palette location is made up of three fields (red, green, and blue) three writes to the Palette Data register are required for each palette location. Palette data is held in a three-field holding register, and the contents are loaded into the palette RAM when all three fields have been filled. The Palette Sequence register controls the Holding Register field (red, green, or blue) selected for access with each write to the Palette Data register.

Two update sequences are possible:

1. Red, green, blue
2. Red, blue, green, no access.

Data is written to the palette by first loading the index, or address, of the first group of three palette-color locations into the non-prefetched Sprite/Palette Index Low register. Because the palette has only 256 locations, the Sprite Index High register is not used. The first color byte is then written to the Palette Data register. This stores the color byte in the Holding Register field indicated by the Palette Sequence register. The Palette Sequence register then increments to point to the next field as determined by the update order.

A second write to the Palette Data register loads the next Holding Register field, and the Palette Sequence register increments again. A third write to the Palette Data register loads the remaining Holding Register field. If update sequence 1 is selected, the palette location is loaded from the holding register and the Palette Sequence register increments again, returning to its starting value. If update sequence 2 is selected, a fourth write to the Palette Data register is necessary before the palette location is loaded. The no-access data is ignored. Update sequence 2 allows the application to take advantage of the word or doubleword access possible with the XGA subsystem. See

"Data Registers (Addresses 21xB to 21xF)" on page 3-42 and "XGA Display Controller Registers" on page 3-30 for more details.

The prefetch function is used to read from the palette. The index or address of the first palette location to be read is loaded into the Sprite/Palette Prefetch Index Low register.

Writing to this register loads the three color fields stored at the location specified in the index register into the palette holding register, then increments the index register.

A subsequent read from the Palette Data register returns the data from the holding register color field, indicated by the Palette Sequence register, and increments the sequence register to point to the next color field. When the last color field, indicated by the Palette Sequence register, is read, the holding register is loaded with the next palette location data, and the index is incremented.

Note: If the subsystem has a monochrome display attached, all of the palette red and blue locations must be loaded with 0's.

The sprite and the palette use the same hardware registers during reading and writing, so any task that is updating either the sprite or palette when an interrupt occurs must save and restore the following registers:

- Sprite/Palette Index Low register (index hex 60)
- Sprite Index High register (index hex 61)
- Palette Sequence register (index hex 66)
- Palette Red Prefetch register (index hex 67)
- Palette Green Prefetch register (index hex 68)
- Palette Blue Prefetch register (index hex 69)
- Sprite Prefetch register (index hex 6B).

Note: The Sprite/Palette Prefetch Index Low register (index hex 62) and Sprite Prefetch Index High register (index hex 63) must not be saved and restored.

Direct Color Mode

In direct color mode the PEL values in the video memory directly specify the displayed color.

The XGA subsystem can display direct color as a 16-bit PEL. The color fields provide the most significant bits of the inputs to the video DACs with the color value.

The bits in the 16-bit direct color data word are allocated to the DAC bits as follows:

Word bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
5R, 6G, 5B	msb		RED		lsb	msb		GREEN			lsb	msb		BLUE		lsb

Figure 3-9. Direct Color Mode Data Word

When selecting this mode, the palette must be loaded with data shown in Figure 3-10 on page 3-28. Only half of the palette should be loaded. Bit 7 of the Border Color register (index hex 55) specifies which half to load. If the Border Color register bit 7 = 0, load the upper half of the palette (locations hex 80 to FF). If the Border Color register bit 7 = 1, load the lower half (locations hex 00 to 7F).

Note: Each palette location is 6 bits wide. The actual bits loaded are from the *most* significant bits of the byte written to the Palette Data register. See "Palette Data Register (Index 65)" on page 3-74 for a description of the register.

The values shown in the following figure are written to the Palette Data register with a byte access command. The most significant bits have been used.

Location (Hex)		Red (Hex)	Green (Hex)	Blue (Hex)
Border Color Bit 7 =				
0	1			
80	0	0	0	0
81	1	0	0	8
82	2	0	0	10
83	3	0	0	18
84	4	0	0	20
.	.	.	.	.
.	.	.	.	.
9E	1E	0	0	F0
9F	1F	0	0	F8
A0	20	0	0	0
A1	21	0	0	8
.	.	.	.	.
.	.	.	.	.
BE	3E	0	0	F0
BF	3F	0	0	F8
C0	40	0	0	0
C1	41	0	0	8
.	.	.	.	.
.	.	.	.	.
DE	5E	0	0	F0
DF	5F	0	0	F8
E0	60	0	0	0
E1	61	0	0	8
.	.	.	.	.
.	.	.	.	.
FE	7E	0	0	F0
FF	7F	0	0	F8

Figure 3-10. XGA Direct Color Palette Load

The values shown in Figure 3-10 have been chosen to ensure future compatibility.

See “Extended Graphics Mode” on page 3-182 and “Direct Color Mode” on page 3-224 for more details on this mode.

Coprocessor Functions

The XGA subsystem coprocessor functions do not work in 16 bits-per-PEL mode. However, the coprocessor can function in 8 bits-per-PEL mode while data is being displayed in 16 bits-per-PEL. As a result, the coprocessor can be used to move data (in PxBlts) from one area of memory to another.

Care should be taken when using any of the logical or arithmetic functions since each operation is performed on only 1 byte of data at a time, not the full 16-bit PEL.

If the coprocessor is used to move data into the Video Display Buffer in 8 bits-per-PEL format while displaying in 16 bits-per-PEL mode, the width of the destination map must be doubled.

See "Direct Color Mode" on page 3-224 for more information.

XGA Display Controller Registers

The display controller registers occupy 16 I/O addresses. The addresses are hex 21x0 through 21xF. The x is the Instance as defined in Figure 3-161 on page 3-153. "Locating the XGA Subsystem" on page 3-171 provides details of locating and using these registers.

An indexed addressing scheme is used to select additional registers. The index of the registers is written to hex 21xA; the data can then be accessed using hex 21xB through 21xF. Because there are multiple addresses for the data port, writes to a single register are achieved in a single 16-bit instruction, the low byte containing the address, and the high byte the data. Registers that need to be accessed repeatedly (sprite data, palette data, and coprocessor save/restore data) are accessed by setting the index correctly, then performing string I/O instructions, either 2 or 4 bytes at a time. See "Data Registers (Addresses 21xB to 21xF)" on page 3-42.

The 16 I/O addresses are assigned as shown in the following figure.

Address (hex)	Function	Page Reference
21x0	Operating Mode register	3-32
21x1	Aperture Control register	3-33
21x2	Reserved	
21x3	Reserved	
21x4	Interrupt Enable register	3-34
21x5	Interrupt Status register	3-36
21x6	Virtual Memory Control register	3-37
21x7	Virtual Memory Interrupt Status register	3-37
21x8	Aperture Index register	3-38
21x9	Memory Access mode	3-39
21xA	Index	3-40
21xB	Data	3-42
21xC	Data	3-42
21xD	Data	3-42
21xE	Data	3-42
21xF	Data	3-42

Figure 3-11. Display Controller Register Addresses

Register Usage Guidelines

Unless specified otherwise, the following are guidelines when using the display controller registers:

- All registers are 8 bits wide.
- Registers can be read and written at the same address or index.
- When registers are read, they return the last written data for all implemented bits.
- Registers are *not* initialized by reset.
- Special reserved register bits must be used as follows:
  - Register bits marked with '–' must be set to 0. These bits are undefined when read and should be masked off if the contents of the register is to be tested.
  - Register bits marked with '#' are reserved and the state of these bits must be preserved. When writing the register, read the register first and change only the bits that must be changed.
- Unspecified registers or registers marked as reserved in the XGA I/O address space are reserved. They must not be written to or read from.
- During a read, the values returned from write-only registers are reserved and unspecified.
- The contents of read-only registers must not be modified.
- Counters must not be relied upon to wrap from the high value to the low value.
- Register fields defined with valid ranges must not be loaded with a value outside the specified range.
- Register field values defined as reserved must not be written.
- The function that all XGA subsystem registers imply is only operative in XGA subsystem modes, even though the registers themselves are still readable and writable in VGA modes.

  Writing to the XGA subsystem registers when in VGA mode may cause the VGA registers to be corrupted.

Direct Access I/O Registers

The following registers are directly addressable in the I/O space hex 21x0 through 21xF.

Operating Mode Register (Address 21x0)

This read/write register has an address of hex 21x0.

7	6	5	4	3	2	1	0
–	–	–	–	RF	DM		

```
 – : Set to 0, Undefined on Read
RF : Coprocessor Register Interface Format
DM : Display Mode
```

Figure 3-12. Operating Mode Register, Address Hex 21x0

The register fields are defined as follows:

RF The Coprocessor Register Interface Format field (bit 3) selects whether the coprocessor registers are arranged in Intel or Motorola format. When set to 0, Intel format is selected. When set to 1, Motorola format is selected. See "Coprocessor Registers" on page 3-118.

DM The Display Mode field (bits 2 – 0) selects between the display modes available. Both VGA and 132-column text modes respond to VGA I/O and memory addresses. When the XGA subsystem is in either of these modes, the addressing of the I/O registers and the video memory can be inhibited.

DM Field (binary)	Display Mode
0 0 0	VGA Mode (address decode disabled)
0 0 1	VGA Mode (address decode enabled)
0 1 0	132-Column Text Mode (address decode disabled)
0 1 1	132-Column Text Mode (address decode enabled)
1 0 0	Extended Graphics Mode
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

Figure 3-13. Display Mode Bit Assignments

Aperture Control Register (Address 21x1)

This read/write register has address of hex 21x1.

7	6	5	4	3	2	1	0
–	–	–	–	–	–	ASL	

```
  – : Set to 0, Undefined on Read
ASL : Aperture Size And Location
```

Figure 3-14. Aperture Control Register, Address Hex 21x1

The register fields are defined as follows:

ASL The Aperture Size and Location field (bits 1, 0) controls a 64KB aperture through which XGA memory can be accessed in system address space. This aperture gives real mode applications and operating systems a means of accessing the XGA video memory. The 64KB area of the XGA video memory accessed by this aperture is selected using the Aperture Index register. By varying the value of the index register, the 64KB aperture is used to access the entire memory contents of the subsystem.

The aperture is controlled as follows:

ASL Field (binary)	Aperture Size and Location
0 0	No 64KB Aperture
0 0	64KB at Address Hex 000A0000
0 1	64KB at Address Hex 000B0000
1 1	Reserved

Figure 3-15. Aperture Size and Location Bit Assignments

The 64KB aperture and a 1MB aperture cannot be used together because they are both paged using the Aperture Index register. See "System Apertures into Video Memory" on page 3-19.

Interrupt Enable Register (Address 21x4)

This read/write register has an address of hex 21x4.

7	6	5	4	3	2	1	0
CC	CR	–	–	–	SC	SP	SB

–	:	Set to 0, Undefined on Read
CC	:	Coprocessor Operation Complete Enable
CR	:	Coprocessor Access Rejected Enable
SC	:	Sprite Display Complete Enable
SP	:	Start Of Picture (End Of Blanking) Enable
SB	:	Start Of Blanking (End Of Picture) Enable

Figure 3-16. Interrupt Enable Register, Address Hex 21x4

The register fields are defined as follows:

CC The Coprocessor Operation Complete Enable field (bit 7) enables and disables the Coprocessor Operation Complete interrupt condition that can be generated by the subsystem. When set to 1, the interrupt is enabled. When set to 0, the interrupt is disabled. The status of the bit in this field has no effect on the interrupt status bits as defined in the Interrupt Status register, but prevents the interrupt condition from causing a system interrupt.

CR The Coprocessor Access Rejected Enable field (bit 6) enables and disables the Coprocessor Access Rejected interrupt condition that can be generated by the subsystem. When set to 1, the interrupt is enabled. When set to 0, the interrupt is disabled. The status of the bit in this field has no effect on the interrupt status bits as defined in the Interrupt Status register, but prevents the interrupt condition from causing a system interrupt.

SC The Sprite Display Complete Enable field (bit 2) enables and disables the Sprite Display Complete interrupt condition that can be generated by the subsystem. When set to 1, the interrupt is enabled. When set to 0, the interrupt is disabled. The status of the bit in this field has no effect on the interrupt status bits as defined in the Interrupt Status register, but prevents the interrupt condition from causing a system interrupt.

SP The Start of Picture (End of Blanking) Enable field (bit 1) enables and disables the Start of Picture interrupt condition that can be generated by the subsystem. When set to 1, the interrupt is enabled. When set to 0, the interrupt is disabled. The status of the bit in this field has no effect on the interrupt status bits as defined in the Interrupt Status register, but prevents the interrupt condition from causing a system interrupt.

SB The Start of Blanking (End of Picture) Enable field (bit 0) enables and disables the Start of Blanking interrupt condition that can be generated by the subsystem. When set to 1, the interrupt is enabled. When set to 0, the interrupt is disabled. The status of the bit in this field has no effect on the interrupt status bits as defined in the Interrupt Status register, but prevents the interrupt condition from causing a system interrupt.

Interrupt Status Register (Address 21x5)

This read/write register has an address of hex 21x5.

7	6	5	4	3	2	1	0
CC	CR	–	–	–	SC	SP	SB

–	:	Set to 0, Undefined on Read
CC	:	Coprocessor Operation Complete Status
CR	:	Coprocessor Access Rejected Status
SC	:	Sprite Display Complete Status
SP	:	Start Of Picture (End Of Blanking) Status
SB	:	Start Of Blanking (End Of Picture) Status

Figure 3-17. Interrupt Status Register, Address Hex 21x5

The register fields are defined as follows:

CC The Coprocessor Operation Complete Status field (bit 7) contains the interrupt status bit that can be generated by the subsystem to reset the Coprocessor Operation Complete interrupt. When read, 1 indicates that the interrupt condition has occurred, and 0 that it has not. Writing a 1 to the bit clears the interrupt condition, while writing a 0 has no effect. See "Programmer's View" on page 3-83 for more information.

CR The Coprocessor Access Rejected Status field (bit 6) contains the interrupt status bit that can be generated by the subsystem to reset the Coprocessor Access Rejected interrupt. When read, 1 indicates that the interrupt condition has occurred, and 0 that it has not. Writing a 1 to the bit clears the interrupt condition, while writing a 0 has no effect. See "Accesses to the Coprocessor During an Operation" on page 3-115 for more information.

SC The Sprite Display Complete Status field (bit 2) contains the interrupt status bit that can be generated by the subsystem to reset the Sprite Display Complete interrupt. When read, 1 indicates that the interrupt condition has occurred, and 0 that it has not. Writing a 1 to the bit clears the interrupt condition, while writing a 0 has no effect. See "Sprite" on page 3-22 for more information.

SP The Start of Picture (End of Blanking) Status field (bit 1) contains the interrupt status bit that can be generated by the subsystem to reset the Start of Picture interrupt. When read, 1 indicates that the interrupt condition has occurred, and 0 that it has not. Writing a 1 to the bit clears the interrupt condition, while writing a 0 has no effect. See "CRT Controller" on page 3-19 (End of blanking) for more information.

SB The Start of Blanking (End of Picture) Status field (bit 0) contains the interrupt status bit that can be generated by the subsystem to reset the Start of Blanking interrupt. When read, 1 indicates that the interrupt condition has occurred, and 0 that it has not. Writing a 1 to the bit clears the interrupt condition, while writing a 0 has no effect. See "CRT Controller" on page 3-19 (Start of blanking) for more information.

Virtual Memory Control Register (Address 21x6)

This read/write register has an address of hex 21x6. Full details of this register are in "Virtual Memory Control Register (I/O Address 21x6)" on page 3-167.

Virtual Memory Interrupt Status Register (Address 21x7)

This read/write register has an address of hex 21x7. Full details of this register are in "Virtual Memory Interrupt Status Register (I/O Address 21x7)" on page 3-169.

Aperture Index Register (Address 21x8)

This read/write register has an address of hex 21x8.

7	6	5	4	3	2	1	0
–	–	Aperture Index					

– : Set to 0, Undefined on Read

Figure 3-18. Aperture Index Register, Address Hex 21x8

The register field is defined as follows:

Aperture Index

The Aperture Index field (bits 5–0) provides address bits to video memory when the aperture in the system address space being used is smaller than the size of video memory installed. They are used to move both the 64KB aperture and the 1MB aperture. All 6 bits are used to move the 64KB aperture in the video memory, with a granularity of 64KB. When moving the 1MB aperture, the granularity is restricted to 1MB and only bits 5 and 4 are used. In this case, the lower order bits must be written with 0′s.

See “System Apertures into Video Memory” on page 3-19 for details on the use of video memory apertures.

The bits are used as follows:

Aperture Size	Index Bits Used
64KB	5–0
1MB	5–4

Figure 3-19. Aperture Index Bit Assignments

Memory Access Mode Register (Address 21x9)

This read/write register has an address of hex 21x9.

7	6	5	4	3	2	1	0
–	–	–	–	PO		PS	

```
 – : Set to 0, Undefined on Read
PO : PEL Order
PS : PEL Size
```

Figure 3-20. Memory Access Mode Register, Address Hex 21x9

The register fields are defined as follows:

PO The PEL Order field (bit 3) controls PEL ordering when the video memory is being accessed by the system (not the coprocessor). Intel or Motorola order can be selected. When set to 0, Intel format is selected. When set to 1, Motorola format is selected.

PS The PEL Size field (bits 2–0) selects the PEL size. The PEL size must be selected because this register is controlling a PEL swapper that converts from the external format specified to the internal format used by the adapter when the PELs are written, and converts back when they are read.

It is important to set this register correctly when accessing video memory with the system processor.

PEL size values are assigned as follows:

PS Field (binary)	PEL Size
0 0 0	1 Bit
0 0 1	2 Bits
0 1 0	4 Bits
0 1 1	8 Bits
1 0 0	16 Bits
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

Figure 3-21. PEL Size Bit Assignments

Index Register (Address 21xA)

This read/write register has an address of hex 21xA.

```
  7    6    5    4    3    2    1    0
┌───────────────────────────────────────┐
│            Register Index             │
└───────────────────────────────────────┘
```

Figure 3-22. Index Register, Address Hex 21xA

The Register Index register selects the indexed Extended Graphics mode register accessed when any address with index hex B through F is read or written. Index values are assigned as shown in the following figure.

Register Index Field (hex)	**Register**
04	Auto-Configuration
0C	Coprocessor Save/Restore Data A
0D	Coprocessor Save/Restore Data B
10	Horizontal Total Low
11	Horizontal Total High
12	Horizontal Display End Low
13	Horizontal Display End High
14	Horizontal Blanking Start Low
15	Horizontal Blanking Start High
16	Horizontal Blanking End Low
17	Horizontal Blanking End High
18	Horizontal Sync Pulse Start Low
19	Horizontal Sync Pulse Start High
1A	Horizontal Sync Pulse End Low
1B	Horizontal Sync Pulse End High
1C	Horizontal Sync Position
1E	Horizontal Sync Position
20	Vertical Total Low
21	Vertical Total High
22	Vertical Display End Low
23	Vertical Display End High
24	Vertical Blanking Start Low
25	Vertical Blanking Start High
26	Vertical Blanking End Low
27	Vertical Blanking End High
28	Vertical Sync Pulse Start Low
29	Vertical Sync Pulse Start High
2A	Vertical Sync Pulse End
2C	Vertical Line Compare Low
2D	Vertical Line Compare High

Note: Undefined index values are reserved.

Figure 3-23. XGA Index Register Assignments (Part I)

Register Index Field (hex)	Register
30	Sprite Horizontal Start Low
31	Sprite Horizontal Start High
32	Sprite Horizontal Preset
33	Sprite Vertical Start Low
34	Sprite Vertical Start High
35	Sprite Vertical Preset
36	Sprite Control
38	Sprite Color 0 Red
39	Sprite Color 0 Green
3A	Sprite Color 0 Blue
3B	Sprite Color 1 Red
3C	Sprite Color 1 Green
3D	Sprite Color 1 Blue
40	Display PEL Map Offset Low
41	Display PEL Map Offset Middle
42	Display PEL Map Offset High
43	Display PEL Map Width Low
44	Display PEL Map Width High
50	Display Control 1
51	Display Control 2
52	Display ID and Comparator
54	Clock Frequency Select 1
55	Border Color
60	Sprite/Palette Index Low
61	Sprite Index High
62	Sprite/Palette Prefetch Index Low
63	Sprite Prefetch Index High
64	Palette Mask
65	Palette Data
66	Palette Sequence
67	Palette Red Prefetch
68	Palette Green Prefetch
69	Palette Blue Prefetch
6A	Sprite Data
6B	Sprite Prefetch
70	Clock Frequency Select 2

Note: Undefined index values are reserved.

Figure 3-24. XGA Index Register Assignments (Part II)

Data Registers (Addresses 21xB to 21xF)

These read/write data registers have addresses of hex 21xB to 21xF. The data registers are used when reading and writing to the register indexed by the Index register (Address 21xA). The read/write operation can be of byte, word, or doubleword size.

To perform a byte write to an indexed register, a single 16-bit cycle to address hex 21xA can be used with the index in the lower byte and the data to be written in the upper byte. For indexed registers requiring successive writes, the index can be loaded using a byte write to address hex 21xA, followed by either a word or a doubleword access to address hex 21xC. Only the byte-wide register selected by the index is updated. Word or doubleword accesses result in two or four byte-wide accesses to the same indexed register.

Indexed Access I/O Registers

See "Index Register (Address 21xA)" on page 3-40 for a figure of the indexed registers.

Auto-Configuration Register (Index 04)

This read-only register has an index of hex 04. *Do not write to* this register.

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	BS

```
 – : Undefined on Read
BS : Bus Size
```

Figure 3-25. Auto-Configuration Register, Index Hex 04

The register field is defined as follows:

BS The Bus Size field (bit 0) indicates whether the subsystem is interfaced to a 16-bit or a 32-bit system interface. When read as 0, the system interface is 16-bit, when read as 1, the system interface is 32-bit.

Coprocessor Save/Restore Data Registers (Index 0C and 0D)

These read/write registers have indexes of hex 0C and 0D. The registers are an image of a port in the coprocessor. See "Coprocessor State Save/Restore" on page 3-116 for a description of their use.

Horizontal Total Registers (Index 10 and 11)

These read/write registers have indexes of hex 10 and 11.

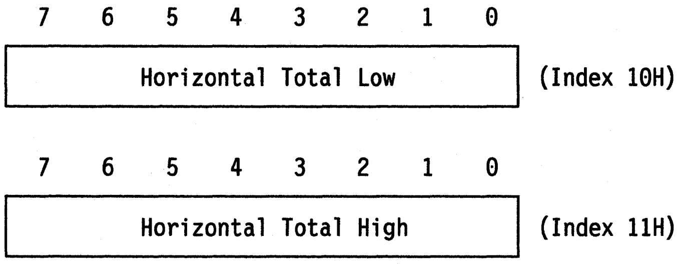


Figure 3-26. Horizontal Total Registers, Indexes Hex 10 and 11

The Horizontal Total Low and Horizontal Total High registers (bits 7 – 0) define the total length of a scan line in units of eight PELs. They *must* be loaded as a 16-bit value in the range hex 0000 to 00FF. Values are assigned as shown in the following figure.

Value (hex)	**Horizontal Total (PELs)**
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-27. Horizontal Total Registers Value Assignments

Horizontal Display End Registers (Index 12 and 13)

These read/write registers have indexes of hex 12 and 13.

```
  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│      Horizontal Display End Low        │  (Index 12H)
└────────────────────────────────────────┘

  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│      Horizontal Display End High       │  (Index 13H)
└────────────────────────────────────────┘
```

Figure 3-28. Horizontal Display End Registers, Indexes Hex 12 and 13

The Horizontal Display End Low and Horizontal Display End High registers (bits 7 – 0) define the position of the end of the active picture area relative to the start of the active picture area in units of eight PELs. They *must* be loaded as a 16-bit value in the range hex 0000 to 00FF. Values are assigned as shown in the following figure.

Value (hex)	Display End (PELs)
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-29. Horizontal Display End Registers Value Assignments

Horizontal Blanking Start Registers (Index 14 and 15)

These read/write registers have indexes of hex 14 and 15.

```
  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│     Horizontal Blanking Start Low      │  (Index 14H)
└────────────────────────────────────────┘

  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│     Horizontal Blanking Start High     │  (Index 15H)
└────────────────────────────────────────┘
```

Figure 3-30. Horizontal Blanking Start Registers, Indexes Hex 14 and 15

The Horizontal Blanking Start Low and Horizontal Blanking Start High registers (bits 7 – 0) define the position of the end of the picture border area relative to the start of the active picture area in units of eight PELs. They *must* be loaded as a 16-bit value in the range hex 0000 to 00FF. Values are assigned as shown in the following figure.

Value (hex)	Blanking Start (PELs)
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-31. Horizontal Blanking Start Registers Value Assignments

Horizontal Blanking End Registers (Index 16 and 17)

These read/write registers have indexes of hex 16 and 17.

```
  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│     Horizontal Blanking End Low        │  (Index 16H)
└────────────────────────────────────────┘

  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│     Horizontal Blanking End High       │  (Index 17H)
└────────────────────────────────────────┘
```

Figure 3-32. Horizontal Blanking End Registers, Indexes Hex 16 and 17

The Horizontal Blanking End Low and Horizontal Blanking End High registers (bits 7 – 0) define the position of the start of the picture border area relative to (after) the start of the active picture area in units of eight PELs. They *must* be loaded as a 16-bit value in the range hex 0000 to 00FF. Values are assigned as shown in the following figure.

Value (hex)	**Blanking End (PELs)**
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-33. Horizontal Blanking End Registers Value Assignments

Horizontal Sync Pulse Start Registers (Index 18 and 19)

These read/write registers have indexes of hex 18 and 19.

```
  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│         HSYNC Pulse Start Low          │  (Index 18H)
└────────────────────────────────────────┘

  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│         HSYNC Pulse Start High         │  (Index 19H)
└────────────────────────────────────────┘
```

Figure 3-34. Horizontal Sync Pulse Start Registers, Indexes Hex 18 and 19

The Horizontal Sync Pulse Start Low and Horizontal Sync Pulse Start High registers (bits 7 – 0) define the position of the start of horizontal sync pulse relative to the start of the active picture area in units of eight PELs. They *must* be loaded as a 16 bit-value in the range hex 0000 to 00FF. Values are assigned as shown in the following figure.

Value (hex)	Horizontal Pulse Start (PELs)
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-35. Horizontal Sync Pulse Start Registers Value Assignments

Horizontal Sync Pulse End Registers (Index 1A and 1B)

These read/write registers have indexes of hex 1A and 1B.

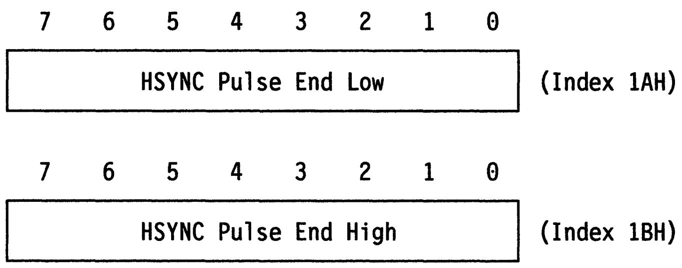


Figure 3-36. Horizontal Sync Pulse End Registers, Indexes Hex 1A and 1B

The Horizontal Sync Pulse End Low and Horizontal Sync Pulse End High registers (bits 7 – 0) define the position of the end of the horizontal sync pulse relative to the start of the active picture area in units of eight PELs. They *must* be loaded as a 16-bit value in the range hex 0000 to 00FF.

This XGA subsystem register is also used in 132-column text mode in place of the VGA End Horizontal Retrace register. In 132-column text mode, each eight-PEL unit is equivalent to one eight-PEL character. Values are assigned as shown in the following figure.

Value (hex)	Horizontal Sync Pulse End (PELs)
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-37. Horizontal Sync Pulse End Registers Value Assignments

Horizontal Sync Pulse Position Registers (Index 1C and 1E)

These write-only registers have indexes of hex 1C and 1E.

7	6	5	4	3	2	1	0	
–	PD		–	–	–	–	–	(Index 1CH)

7	6	5	4	3	2	1	0	
–	–	–	–	–	PD		–	(Index 1EH)

```
 - : Set to 0
PD : Sync Pulse Delay
```

Figure 3-38. Horizontal Sync Pulse Position Registers (Index 1C and 1E)

The register field is defined as follows:

PD The Sync Pulse Delay field (bits 6, 5 or bits 2, 1) allows the 'horizontal sync' (HSYNC) signal to be delayed by up to four PELs. The same value *must* be written to both registers, as shown in the following figure.

PD Field (binary)	**Sync Pulse Delay in PELs**
0 0	0
0 1	Reserved
1 0	4
1 1	Reserved

Figure 3-39. Horizontal Sync Pulse Delay Bit Assignments

These XGA subsystem registers are also used in 132-column text mode in place of the HRD field in the VGA End Horizontal Retrace register.

Vertical Total Registers (Index 20 and 21)

These read/write registers have indexes of hex 20 and 21.

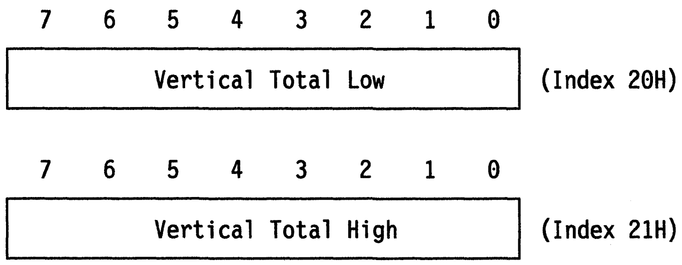


Figure 3-40. Vertical Total Registers, Indexes Hex 20 and 21

The Vertical Total Low and Vertical Total High registers (bits 7 – 0) define the total length of a frame in units of one scan line. They *must* be written as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	Total Length (Scan Lines)
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-41. Vertical Total Registers Value Assignments

Vertical Display End Registers (Index 22 and 23)

These read/write registers have indexes of hex 22 and 23.

```
  7   6   5   4   3   2   1   0
┌───────────────────────────────┐
│   Vertical Display End Low    │  (Index 22H)
└───────────────────────────────┘

  7   6   5   4   3   2   1   0
┌───────────────────────────────┐
│   Vertical Display End High   │  (Index 23H)
└───────────────────────────────┘
```

Figure 3-42. Vertical Display End Registers, Indexes Hex 22 and 23

The Vertical Display End Low and Vertical Display End High registers (bits 7 – 0) define the position of the end of the active picture area relative to the start of the active picture area in units of one scan line. They *must* be written as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	Display End (Scan Lines)
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-43. Vertical Display End Registers Value Assignments

Vertical Blanking Start Registers (Index 24 and 25)

These read/write registers have indexes of hex 24 and 25.

```
   7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│     Vertical Blanking Start Low        │  (Index 24H)
└────────────────────────────────────────┘

   7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│     Vertical Blanking Start High       │  (Index 25H)
└────────────────────────────────────────┘
```

Figure 3-44. Vertical Blanking Start Registers, Indexes Hex 24 and 25

The Vertical Blanking Start Low and Vertical Blanking Start High registers (bits 7 – 0) define the position of the end of the picture border area relative to the start of the active picture area in units of one scan line. They *must* be loaded as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	Border End (Scan Lines) Blanking Start
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-45. Vertical Blanking Start Registers Value Assignments

Vertical Blanking End Registers (Index 26 and 27)

These read/write registers have indexes of hex 26 and 27.

```
  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│       Vertical Blanking End Low        │  (Index 26H)
└────────────────────────────────────────┘

  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│       Vertical Blanking End High       │  (Index 27H)
└────────────────────────────────────────┘
```

Figure 3-46. Vertical Blanking End Registers, Indexes Hex 26 and 27

The Vertical Blanking End Low and Vertical Blanking End High registers (bits 7 – 0) define the position of the start of the picture border area relative to the start of the active picture area in units of one scan line. They *must* be loaded as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	**Border Start (Scan Lines) Blanking End**
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-47. Vertical Blanking End Registers Value Assignments

Vertical Sync Pulse Start Registers (Index 28 and 29)

These read/write registers have indexes of hex 28 and 29.

```
  7    6    5    4    3    2    1    0
┌───────────────────────────────────────┐
│         VSYNC Pulse Start Low         │  (Index 28H)
└───────────────────────────────────────┘

  7    6    5    4    3    2    1    0
┌───────────────────────────────────────┐
│         VSYNC Pulse Start High        │  (Index 29H)
└───────────────────────────────────────┘
```

Figure 3-48. Vertical Sync Pulse Start Registers, Indexes Hex 28 and 29

The Vertical Sync Pulse Start Low and Vertical Sync Pulse Start High registers (bits 7 – 0) define the position of the start of the vertical sync pulse relative to the start of the active picture area in units of one scan line. They *must* be loaded as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	Sync Pulse Start (Scan Lines)
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-49. Vertical Sync Pulse Start Registers Value Assignments

Vertical Sync Pulse End Register (Index 2A)

This read/write register has an index of hex 2A.

```
  7    6    5    4    3    2    1    0
┌───────────────────────────────────────┐
│            VSYNC Pulse End            │
└───────────────────────────────────────┘
```

Figure 3-50. Vertical Sync Pulse End Register, Index Hex 2A

The Vertical Sync Pulse End register (bits 7 – 0) defines the position of the end of the vertical sync pulse. The value loaded is the least significant byte of a 16-bit value that defines the end of the vertical sync pulse relative to the start of the active picture area in units of one scan line. The vertical sync end position *must* be within 31 scan lines of the vertical sync start position.

Note: Before setting the Operating Mode register (address 21x0) into VGA or 132-column text mode, bit 5 of this register must be set to 1.

This register may not return the value written, but the returned value is valid for save/restore operations.

Vertical Line Compare Registers (Index 2C and 2D)

These read/write registers have indexes of hex 2C and 2D.

```
  7    6    5    4    3    2    1    0
┌─────────────────────────────────────────┐
│       Vertical Line Compare Low         │  (Index 2CH)
└─────────────────────────────────────────┘

  7    6    5    4    3    2    1    0
┌─────────────────────────────────────────┐
│       Vertical Line Compare High        │  (Index 2DH)
└─────────────────────────────────────────┘
```

Figure 3-51. Vertical Line Compare Registers, Indexes Hex 2C and 2D

The Vertical Line Compare Low and Vertical Line Compare High registers (bits 7 – 0) define the position of the end of the scrollable picture area relative to the start of the active picture area in units of one scan line. They *must* be loaded as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	Scrollable End (scan lines)
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-52. Vertical Line Compare Registers Value Assignments

Sprite Horizontal Start Registers (Index 30 and 31)

These read/write registers have indexes of hex 30 and 31.

7	6	5	4	3	2	1	0	
Sprite Horizontal Start Low								(Index 30H)

7	6	5	4	3	2	1	0	
Sprite Horizontal Start High								(Index 31H)

Figure 3-53. Sprite Horizontal Start Registers, Indexes Hex 30 and 31

The Sprite Horizontal Start Low and Sprite Horizontal Start High registers (bits 7 – 0) define the position of the start of the sprite relative to the start of the active picture area in PELs. They *must* be loaded with a 16-bit value in the range hex 0000 to 07FF. See "Sprite Positioning" on page 3-24. Values are assigned as shown in the following figure.

Value (hex)	**Sprite Start (PELs)**
0000	0
0001	1
0002	2
.	
.	
07FF	2047

Figure 3-54. Sprite Horizontal Start Registers Value Assignments

Sprite Horizontal Preset Register (Index 32)

This read/write register has an index of hex 32.

7	6	5	4	3	2	1	0
–	–	Sprite Horizontal Preset					

– : Set to 0, Undefined on Read

Figure 3-55. Sprite Horizontal Preset, Index Hex 32

The register fields are defined as follows:

Sprite Horizontal Preset
The Sprite Horizontal Preset field (bits 5–0) defines the horizontal position within the 64 x 64-PEL sprite area where the sprite starts. The sprite always ends at position 63 (it does not wrap). See "Sprite Positioning" on page 3-24. Values are assigned as shown in the following figure.

Value (hex)	**Sprite Start (PELs)**
00	0
01	1
02	2
.	
.	
3F	63

Figure 3-56. Sprite Horizontal Preset Value Assignments

Sprite Vertical Start Registers (Index 33 and 34)

These read/write registers have indexes of hex 33 and 34.

```
 7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│       Sprite Vertical Start Low        │  (Index 33H)
└────────────────────────────────────────┘

 7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│       Sprite Vertical Start High       │  (Index 34H)
└────────────────────────────────────────┘
```

Figure 3-57. Sprite Vertical Start Registers, Indexes Hex 33 and 34

The Sprite Vertical Start Low and Sprite Vertical Start High registers (bits 7 – 0) define the position of the start of the sprite relative to the start of the active picture area in units of one scan line. They *must* be loaded with a 16-bit value in the range hex 0000 to 07FF. See "Sprite Positioning" on page 3-24. Values are assigned as shown in the following figure.

Value (hex)	Sprite Start (Scan Lines)
0000	0
0001	1
0002	2
.	
.	
07FF	2047

Figure 3-58. Sprite Vertical Start Registers Value Assignments

Sprite Vertical Preset Register (Index 35)

This read/write register has an index of hex 35.

7	6	5	4	3	2	1	0
–	–	Sprite Vertical Preset					

– : Set to 0, Undefined on Read

Figure 3-59. Sprite Vertical Preset, Index Hex 35

The register fields are defined as follows:

Sprite Vertical Preset
The Sprite Vertical Preset field (bits 5 – 0) defines the vertical position within the 64 x 64-PEL sprite area where the sprite starts. The sprite always ends at position 63 (it does not wrap). See "Sprite Positioning" on page 3-24. Values are assigned as shown in the following figure.

Value (hex)	Sprite Start (PELs)
00	0
01	1
02	2
.	
.	
3F	63

Figure 3-60. Sprite Vertical Preset Value Assignments

Sprite Control Register (Index 36)

This read/write register has an index of hex 36.

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	SC

```
 – : Set to 0, Undefined on Read
SC : Sprite Control
```

Figure 3-61. Sprite Control Register, Index Hex 36

The register fields are defined as follows:

SC The Sprite Control field (bit 0) controls the visibility of the sprite. When set to 1, the sprite appears on the screen at the location controlled by the sprite position registers. When set to 0, a sprite is not displayed. This bit must be set to 0 before any attempt is made to access the sprite image in the sprite buffer, otherwise the sprite buffer contents are corrupted.

Sprite Color Registers (Index 38 – 3D)

These read/write registers have indexes of hex 38 through 3D.

7	6	5	4	3	2	1	0	
Sprite Color 0 Red								(Index 38H)

7	6	5	4	3	2	1	0	
Sprite Color 0 Green								(Index 39H)

7	6	5	4	3	2	1	0	
Sprite Color 0 Blue								(Index 3AH)

7	6	5	4	3	2	1	0	
Sprite Color 1 Red								(Index 3BH)

7	6	5	4	3	2	1	0	
Sprite Color 1 Green								(Index 3CH)

7	6	5	4	3	2	1	0	
Sprite Color 1 Blue								(Index 3DH)

Figure 3-62. Sprite Color Registers, Indexes Hex 38 – 3D

The Sprite Color registers (bits 7 – 0) define the red, green, and blue components of the PELs displayed when the sprite data for those PELs selects color 0 or color 1. These colors are passed directly to the DACs, not through the palette, and must be programmed to give the actual color required.

Note: Only the 6 most-significant bits of these registers are used.

Display PEL Map Offset Registers (Index 40 – 42)

These read/write registers have indexes of hex 40 through 42.

7	6	5	4	3	2	1	0	
Display PEL Map Offset Low								(Index 40H)

7	6	5	4	3	2	1	0	
Display PEL Map Offset Middle								(Index 41H)

7	6	5	4	3	2	1	0	
Display PEL Map Offset High								(Index 42H)

Figure 3-63. Display PEL Map Offset Registers, Indexes Hex 40 – 42

The Display PEL Map Offset registers (bits 7 – 0) define the address of the start of the visible portion of the video buffer in units of 8 bytes. They *must* be loaded as a single value in the range hex 00000 to 1FFFF. See "Scrolling" on page 3-21. Values are assigned as shown in the following figure.

Value (hex)	Display PEL Map Offset (bytes)
00000	0
00001	8
00002	16
.	
.	
1FFFF	1048568

Figure 3-64. Display PEL Map Offset Registers Value Assignments

Display PEL Map Width Registers (Index 43 and 44)

These read/write registers have indexes of hex 43 and 44.

```
 7   6   5   4   3   2   1   0
┌───────────────────────────────┐
│   Display PEL Map Width Low   │ (Index 43H)
└───────────────────────────────┘

 7   6   5   4   3   2   1   0
┌───────────────────────────────┐
│   Display PEL Map Width High  │ (Index 44H)
└───────────────────────────────┘
```

Figure 3-65. Display PEL Map Width Registers, Indexes Hex 43 and 44

The Display PEL Map Width Low and Display PEL Map Width High registers (bits 7 – 0) define the width of the display PEL map in units of 8 bytes. They *must* be loaded as a single value in the range hex 000 to 3FF. See "Scrolling" on page 3-21. Values are assigned as shown in the following figure.

Value (hex)	Display PEL Map Width (bytes)
000	0
001	8
002	16
.	
.	
7FF	16376

Figure 3-66. Display PEL Map Width Registers Value Assignments

Display Control 1 Register (Index 50)

This read/write register has an index of hex 50.

7	6	5	4	3	2	1	0
SP		#	VE	SO	1	DB	

```
 # : Preserve Value Read When Writing
 1 : Set to 1, Undefined on Read
SP : SYNC Polarity
VE : Video Extension
SO : Display Scan Order
DB : Display Blanking
```

Figure 3-67. Display Control 1 Register, Index Hex 50

The register fields are defined as follows:

SP The SYNC Polarity field (bits 7, 6) value is assigned as shown in the following figure.

SP Field (binary)	**Vertical**	**Horizontal**	**Lines**
0 0	+	+	768
0 1	+	−	400
1 0	−	+	350
1 1	−	−	480

Figure 3-68. Sync Polarity Bit Assignments

VE The Video Extension field (bit 4) must be set to 1 (enabled) when the subsystem is in VGA mode. It must be be set to 0 (disabled) if the subsystem is in Extended Graphics or 132-column text mode.

SO The Display Scan Order field (bit 3) determines whether the display scan order is interlaced. When set to 0, the display scan order is not interlaced. When set to 1, the display scan order is interlaced.

DB The Display Blanking field (bits 1, 0) value is assigned as shown in the following figure.

DB Field (binary)	Display Blanking
00	Display Blanked, CRT Controller Reset
01	Display Blanked, Prepare for Reset
10	Reserved
11	Normal Operation

Figure 3-69. Display Blanking Bit Assignments

When resetting the CRT controller, the display blanking bits must be set to 01 (prepare for reset) first, followed by 00 (CRT controller reset).

Display Control 2 Register (Index 51)

This read/write register has an index of hex 51.

7	6	5	4	3	2	1	0
VSF		HSF		-	PS		

```
  - : Set to 0, Undefined on Read
VSF : Vertical Scale Factor
HSF : Horizontal Scal Factor
 PS : PEL Size
```

Figure 3-70. Display Control 2 Register, Index Hex 51

The register fields are defined as follows:

VSF The Vertical Scale Factor field (bits 7, 6) controls how many times each line is replicated. Values are assigned as shown in the following figure.

VSF Field (binary)	**Vertical Scale Factor**
0 0	1
0 1	2
1 0	4
1 1	Reserved

Figure 3-71. Vertical Scale Factor Bit Assignments

HSF The Horizontal Scale Factor field (bits 5, 4) controls how many times each PEL is replicated horizontally. Values are assigned as shown in the following figure.

HSF Field (binary)	**Horizontal Scale Factor**
0 0	1
0 1	2
1 0	4
1 1	Reserved

Figure 3-72. Horizontal Scale Factor Bit Assignments

PS The PEL Size field (bits 2 – 0) defines the PEL size (for the serializer, palette, and DAC), and the display scale factors (horizontal and vertical). Values are assigned as shown in the following figure.

PS Field (binary)	PEL Size
0 0 0	1 Bit
0 0 1	2 Bits
0 1 0	4 Bits
0 1 1	8 Bits
1 0 0	16 Bits (Direct Color Mode)
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

Figure 3-73. Display Control 2 Register PEL Size Bit Assignments

Display ID and Comparator Register (Index 52)

This read-only register has an index of hex 52. *Do not write to* this register.

7	6	5	4	3	2	1	0
BD	GD	RD	–	DT			

–	:	Undefined on Read
BD	:	Blue DAC Comparator Status
GD	:	Green DAC Comparator Status
RD	:	Red DAC Comparator Status
DT	:	Display Type

Figure 3-74. Display ID and Comparator, Index Hex 52

The register fields are defined as follows:

BD The Blue DAC Comparator Status field (bit 7) indicates the state of the blue DAC output. When read as 1, the blue DAC output is high; when read as 0, the blue DAC output is low.

GD The Green DAC Comparator Status field (bit 6) indicates the state of the green DAC output. When read as 1, the green DAC output is high; when read as 0, the green DAC output is low.

RD The Red DAC Comparator Status field (bit 5) indicates the state of the red DAC output. When read as 1, the red DAC output is high; when read as 0, the red DAC output is low.

DT The Display Type field (bits 3–0) indicates the type of display attached. Bit values are defined by displays.

Clock Frequency Select 1 Register (Index 54)

This read/write register has an index of hex 54.

7	6	5	4	3	2	1	0
–	–	–	–	CS1		VCS	

–	:	Set to 0, Undefined on Read
CS1	:	Clock Select 1
VCS	:	Video Clock Scale Factor

Figure 3-75. Clock Frequency Selector Register, Index Hex 54

The Clock Frequency Select 1 register must be used in conjunction with the Clock Frequency Select 2 register. It is defined under "Clock Frequency Select 2 Register (Index 70)" on page 3-78.

Border Color Register (Index 55)

This read/write register has an index of hex 55.

7	6	5	4	3	2	1	0
Border Color							

Figure 3-76. Border Color Register, Index Hex 55

The Border Color register (bits 7 – 0) holds the border color palette index, which is the index of the palette location selected to be displayed in the picture border area of the display.

The inverse of bit 7 is used for palette address bit 7 in direct color mode. See "Direct Color Mode" on page 3-27.

Sprite/Palette Index Registers (Index 60 and 61)

These read/write registers have indexes of hex 60 and 61.

```
  7    6    5    4    3    2    1    0
┌───────────────────────────────────────┐
│        Sprite/Palette Index Low       │  (Index 60H)
└───────────────────────────────────────┘

  7    6    5    4    3    2    1    0
┌───────────────────────────────────────┐
│           Sprite Index High           │  (Index 61H)
└───────────────────────────────────────┘
```

Figure 3-77. Sprite/Palette Index Registers, Indexes Hex 60 and 61

The Sprite/Palette Index Low and Sprite Index High registers (bits 7 – 0) specify the index when writing to the sprite or the palette. See "Sprite Buffer Accesses" on page 3-23 and "Palette Accesses" on page 3-25 for details of these registers.

The Sprite/Palette Index Low register is used for the 256 locations of the palette that are available. It can be loaded with any palette index value in the range hex 00 to FF.

The Sprite/Palette Index Low and the Sprite Index High registers are both used to access the sprite. The registers can be loaded with any sprite index value in the range hex 0000 to 3FFF.

Accessing these registers does not cause any action other than loading or returning the value of the next index.

The registers must be saved, and subsequently restored, by any interrupting task that uses the palette or sprite registers.

Sprite/Palette Prefetch Index Registers (Index 62 and 63)

These read/write registers have indexes of hex 62 and 63.

```
  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│   Sprite/Palette Prefetch Index Low    │  (Index 62H)
└────────────────────────────────────────┘

  7    6    5    4    3    2    1    0
┌────────────────────────────────────────┐
│       Sprite Prefetch Index High       │  (Index 63H)
└────────────────────────────────────────┘
```

Figure 3-78. Sprite/Palette Prefetch Index Registers, Indexes 62 and 63

The Sprite/Palette Prefetch Index Low and Sprite Prefetch Index High registers (bits 7 – 0) specify the index when reading from the sprite or the palette. See "Sprite Buffer Accesses" on page 3-23 and "Palette Accesses" on page 3-25 for details of these registers.

When reading from the palette, the Sprite/Palette Prefetch Index Low register must be used. Writing the Sprite/Palette Prefetch Index Low register also causes the palette prefetch registers to be loaded, and the index value to be incremented.

When reading from the sprite, use either the Sprite/Palette Prefetch Index Low register or the Sprite Prefetch Index High register. Writing to either register also causes the sprite prefetch registers to be loaded, and the index value to be incremented as a single value.

These registers must *not* be saved and subsequently restored in hardware task switches.

Palette Mask Register (Index 64)

This read/write register has an index of hex 64.

7	6	5	4	3	2	1	0
Palette Mask							

Figure 3-79. Palette Mask Register, Index Hex 64

The contents of the Palette Mask register (bits 7 – 0) are ANDed with each display memory PEL value, and the result is used to index the palette.

Palette Data Register (Index 65)

This read/write register has an index of hex 65.

7	6	5	4	3	2	1	0
Palette Data							

Figure 3-80. Palette Data Register, Index Hex 65

The Palette Data register (bits 7 – 0) is an image of the currently selected palette RAM location. The data returned on read may not be that last written because of the selection mechanism described in "Palette Accesses" on page 3-25.

For monochrome displays, all of the palette red and blue locations *must* be loaded with 0's.

Note: Only the 6 most-significant bits of this register are used.

Palette Sequence Register (Bits 2 – 0 only) (Index 66)

This read/write register has an index of hex 66.

7	6	5	4	3	2	1	0
–	–	–	–	–	CO	PC	

```
 - : Set to 0, Undefined on Read
CO : Color Order
PC : Palette Color
```

Figure 3-81. Palette Sequence Register, Index Hex 66

The register fields are defined as follows:

CO The Color Order field (bit 2) defines the sequence to be followed for selecting the red, green, and blue elements during successive Palette Data register accesses. The color order is shown in the following figure.

CO Field (binary)	**Color Order**
0	R,G,B,R,G,B,...
1	R,B,G,x,R,B,G,x,..

Note: x = discarded data.

Figure 3-82. Palette Sequence Register Color Order Bit Assignment

PC The Palette Color field (bits 1, 0) defines which of the red, green, or blue elements of the currently selected palette location is the current one for the Palette Data register. The palette color selection is shown in the following figure.

PC Field (binary)	**Color**
0 0	R
0 1	G
1 0	B
1 1	x

Note: x = discarded data.

Figure 3-83. Palette Sequence Register Palette Color Bit Assignments

See “Palette Accesses” on page 3-25 for more information.

Palette Red Prefetch Register (Index 67)

This read/write register has an index of hex 67.

7	6	5	4	3	2	1	0
Palette Red Prefetch							

Figure 3-84. Palette Red Prefetch Register, Index Hex 67

The Palette Red Prefetch register (bits 7 – 0) is not used for any normal function but must be saved, and subsequently restored, by any interrupting code that uses the sprite or palette registers.

Palette Green Prefetch Register (Index 68)

This read/write register has an index of hex 68.

7	6	5	4	3	2	1	0
Palette Green Prefetch							

Figure 3-85. Palette Green Prefetch Register, Index Hex 68

The Palette Green Prefetch register (bits 7 – 0) is not used for any normal function but must be saved, and subsequently restored, by any interrupting code that uses the sprite or palette registers.

Palette Blue Prefetch Register (Index 69)

This read/write register has an index of hex 69.

7	6	5	4	3	2	1	0
Palette Blue Prefetch							

Figure 3-86. Palette Blue Prefetch Register, Index Hex 69

The Palette Blue Prefetch register (bits 7 – 0) is not used for any normal function but must be saved, and subsequently restored, by any interrupting code that uses the sprite or palette registers.

Sprite Data Register (Index 6A)

This read/write register has an index of hex 6A.

7	6	5	4	3	2	1	0
Sprite Data							

Figure 3-87. Sprite Data Register, Index Hex 6A

The Sprite Data register (bits 7 – 0) is an image of the currently selected sprite buffer location. The data returned on read may not be that last written because of the selection mechanism described in "Sprite Buffer Accesses" on page 3-23.

When used for writing sprite data, the sprite PELs are Intel format packed PELs.

Sprite Prefetch Register (Index 6B)

This read/write register has an index of hex 6B.

7	6	5	4	3	2	1	0
Sprite Prefetch							

Figure 3-88. Sprite Prefetch Register, Index Hex 6B

The Sprite Prefetch register (bits 7 – 0) is not used for any normal function but must be saved, and subsequently restored, by any interrupting code that uses the sprite or palette registers.

Clock Frequency Select 2 Register (Index 70)

This read/write register has an index of hex 70. It is used with the Clock Frequency Select 1 Register (index hex 54) for clock selection.

7	6	5	4	3	2	1	0	
–	–	–	–	CS1		VCS		(Index 54H)

7	6	5	4	3	2	1	0	
CS2	–	–	–	–	–	–	–	(Index 70H)

–	:	Set to 0, Undefined on Read
CS1	:	Clock Select 1
CS2	:	Clock Select 2
VCS	:	Video Clock Scale Factor

Figure 3-89. Clock Frequency Select Registers

The clock frequency select registers fields are defined as follows:

CS2 The Clock Select 2 field (bit 7 of the Clock Frequency Select 2 register) is used in conjunction with the Clock Frequency Select 1 register (index hex 54). See the description of the Clock Select 1 field to learn how it is used.

CS1 The Clock Select 1 field (bits 3, 2 of the Clock Frequency Select 1 register) must be used in conjunction with the Clock Frequency Select 2 register. Clock selection is shown in the following figure.

CS2 Field (binary)	CS1 Field (binary)	Selected Clock
0	0 0	VGA 8-PEL Character Mode and 640 x 480 Graphics Mode Clock
0	0 1	VGA 9-PEL Character Mode Clock
0	1 0	Clock Sourced from Video Extension Interface
0	1 1	1024 x 768 Graphics Mode Clock
1	0 0	132-Column Text Mode Clock
1	0 1	Reserved
1	1 0	Reserved
1	1 1	Reserved

Figure 3-90. Clock Selected Bit Assignments

Note: If the 132-column text mode clock is selected, the CS2 field must be set to 0 before any VGA or 640 x 480 graphics are selected. See “Setting the XGA Subsystem Mode” on page 3-182 for details of mode switching.

VCS The Video Clock Scale Factor field (bits 1, 0 of the Clock Frequency Select 1 register) controls the divide ratio of the selected video clock before it is used by the CRT controller. The operation of the video clock scale factor is invisible to the programmer, but it must be set as shown for correct operation of the hardware.

VCS Field (binary)	Video Clock Scale Factor	Mode
0 0	1	VGA and 640 x 480 Graphics Modes
0 1	2	1024 x 768 Graphics and 132-Column Text Modes
1 0	Reserved	
1 1	Reserved	

Figure 3-91. Video Clock Scale Factor Bit Assignments

Coprocessor Description

The XGA coprocessor provides autonomous drawing functions for the video subsystem. Autonomous drawing functions means that the coprocessor draws into memory (either video memory or system memory) independently of the system microprocessor, while the system microprocessor is performing some other operation.

The coprocessor supports 1, 2, 4, or 8 bits-per-PEL. See "Direct Color Mode" on page 3-224 for details of using the coprocessor when displaying in 16 bits-per-PEL (direct color) mode.

The execution of an operation using the coprocessor involves the following steps:

1. The system microprocessor sets up the coprocessor registers to perform a particular operation.
2. The system microprocessor writes to the PEL Operations register to start the coprocessor operation.
3. The coprocessor performs the drawing operation. The system microprocessor can be performing some other function at this time.
4. The coprocessor completes the drawing operation, informs the system microprocessor, and becomes idle.
5. The process is repeated.

The coprocessor operates on PELs within PEL maps. A PEL map is an area of memory at a given address with a defined height, width, and PEL format (see "PEL Maps" on page 3-85).

PELs from a source are combined with PELs from a destination under the control of a pattern and mask, and the result is written back to the destination.

After each access, the source, destination, pattern, and mask addresses are updated according to the function being performed, and the operation is repeated until a programmed limit is encountered.

The drawing operation can be a PEL block transfer (PxBlt), Bresenham line draw, or draw and step.

The function performed to combine the source and destination data can be a logical or arithmetic operation. One of two possible operations is selected for each PEL by the value of the corresponding pattern PEL. A mask PEL for each PEL protects the destination from update.

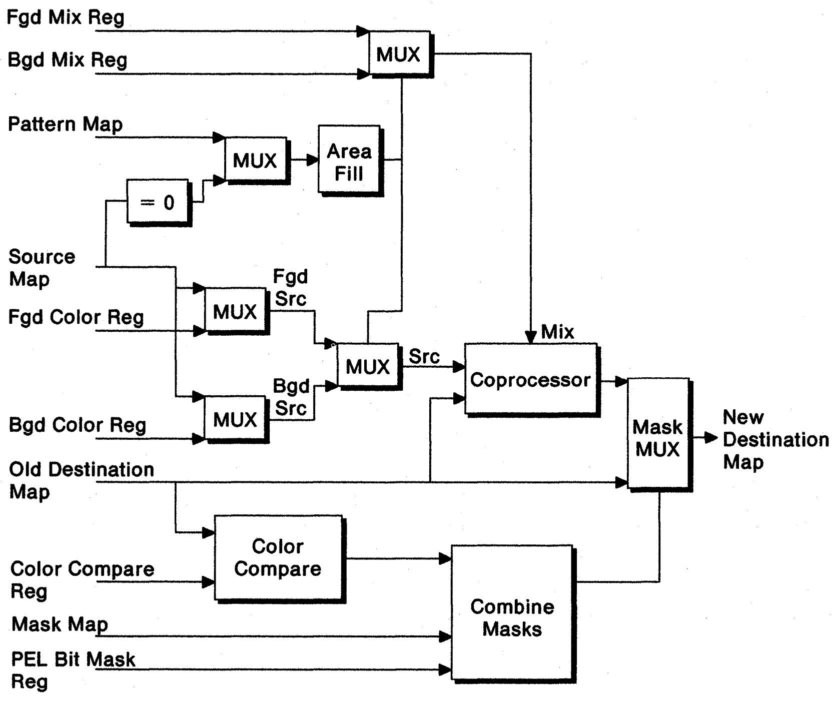
Pattern data can be generated automatically from source data by detecting PELs with a value of 0.

A color compare function allows the modifying of the destination PEL to be dependent on the value of the destination PEL, compared to a programmable value.

Three general purpose PEL maps can be defined in memory. Each map has a defined start address, width and height in PELs, and number of bits-per-PEL. Source, pattern, and destination data can reside in any combination of these maps. There is also a mask map with its defined start address, width, height, and format. Mask data is always taken from this map.

Source, pattern, and destination data are each addressed by unique X,Y pointers. Mask data is addressed by the destination X,Y pointers (see Figure 3-93 on page 3-86). If the source or pattern X,Y pointers move outside the defined extremities of their PEL maps, they are reset automatically to wrap round to the opposite side of the PEL map. If the destination X,Y pointers move outside the extremities of the destination map, update of the destination map is inhibited until the X,Y pointers move back inside the map.

Figure 3-92 on page 3-82 represents the coprocessor graphics data flow for the passage of one PEL. In reality, multiple PELs are processed in one cycle.



Fgd	- Foreground	MUX	- Multiplexer
Bgd	- Background	Reg	- Register
Src	- Source		

Figure 3-92. Coprocessor Data Flow

Programmer's View

An operation is defined as the execution of a single PxBlt, line draw, or draw and step function.

An operation is set up by first loading the registers of the coprocessor with appropriate data, such as X,Y coordinates, function mixes, and dimensions. Then the operation is initiated by writing to the PEL Operations register. This defines the flow of data in the operation and starts the operation. The coprocessor then executes the operation and completes it when some programmed limit is reached.

There is one exception to this sequence of initiating operations: the draw and step function (see the section "Draw and Step" on page 3-95).

The XGA can be programmed to inform the system microprocessor of the completion of an operation using a system interrupt. This interrupt is called the coprocessor-operation-complete interrupt. An enable bit and status bit exist for this interrupt in the "Interrupt Enable Register (Address 21x4)" on page 3-34 and "Interrupt Status Register (Address 21x5)" on page 3-36.

A mechanism is provided to let the system microprocessor suspend or terminate an operation before it is completed. The suspension of operations is required to allow task switches, while termination of operations can be used to recover from errors.

PEL Formats

The coprocessor can manipulate images with 1, 2, 4, or 8 bits-per-PEL. It manipulates packed-PEL data, so each data doubleword (32 bits) contains 32, 16, 8, or 4 PELs respectively.

The PELs can be in Motorola or Intel format. See Figure 3-3 on page 3-17 and Figure 3-4 on page 3-18 for Intel and Motorola formats.

Each PEL map manipulated by the coprocessor can be defined as either Motorola or Intel format. If the destination map has a different format than the source, pattern, or mask maps, the coprocessor automatically translates between the two formats.

Motorola or Intel format is controlled by a bit in the PEL Map Format register.

PEL Fixed and Variable Data

When executing an operation, the coprocessor reads source, pattern, and mask data, and reads and writes destination data. The source, pattern, and mask data can be fixed throughout the operation, or it can vary from PEL to PEL.

If fixed data is used, it is written to the relevant fixed data register in the coprocessor before the operation is started (Foreground Color and Background Color registers).

If variable data is required, the data is read from memory by the coprocessor during the operation. The coprocessor only allows variable data from memory, and does not let the system unit system microprocessor supply variable data.

The Coprocessor View of Memory

To the programmer, the coprocessor treats video memory and system memory the same. Data can be moved between system memory and video memory by defining PEL maps at the appropriate addresses.

Accesses to the XGA video memory are faster than accesses to system memory.

The coprocessor can address all of the video memory.

The Video Memory Base Address register and Instance indicate the base address where the video memory appears in system address space. This base address is on a 4MB address boundary. The coprocessor assumes that the whole 4MB of address space above this boundary is reserved for its own video memory. All addresses outside this 4MB block are treated as system memory. See "POS Register 4 (Base + 4)" on page 3-154.

"Direct Access to Video Memory" on page 3-18 describes video memory addressing.

PEL Maps

PEL Maps A, B, and C (General Maps)

The coprocessor defines three general purpose PEL maps in memory, called PEL maps A, B, and C. Each map is defined by four registers:

PEL Map Base Pointer	Specifies the linear start address of the map in memory.
PEL Map Width	Specifies the width of the map in PELs. The value programmed must be one less than the required width.
PEL Map Height	Specifies the height of the map in PELs. The value programmed must be one less than the required height.
PEL Map Format	Specifies the number of bits-per-PEL of the map, and whether the PELs are stored in Motorola or Intel format.

Source, pattern, and destination data reside in any of PEL maps A, B, or C, determined by the contents of the PEL Operations register.

These maps may be defined as any arbitrary size up to 4096 by 4096 PELs. Individual PELs within the maps are addressed using X,Y pointers. See "X and Y Pointers" on page 3-87.

PEL maps can be located in video memory and in system memory.

There are two restrictions on map usage: the source and destination maps must have the same number of bits-per-PEL, and the pattern map must be 1 bit-per-PEL.

PEL Map M (Mask Map)

In addition to the three general purpose maps, the coprocessor defines a mask map. This map is closely related to the destination map. It protects the destination from update on a PEL-by-PEL basis and can be used to provide a scissoring-type function on any arbitrary shaped area. See “Scissoring with the Mask Map” on page 3-90.

The mask map is described by a set of registers similar to the general purpose PEL maps A, B, and C, but it is fixed at 1 bit-per-PEL.

The mask map differs from the source, pattern, and destination maps as follows:

- The mask map uses the destination X and Y pointers.
- The position of the mask map origin relative to the destination is defined by the mask map origin X and Y offsets.

See “X and Y Pointers” on page 3-87 for more information.

Map Origin

The origin of a PEL map is the point where X = 0 and Y = 0.

The coprocessor defines the origin of all its PEL maps as being at the top left corner of the map. The direction of increasing X is to the right; the direction of increasing Y is downward. Figure 3-93 illustrates the X,Y addressing of an XGA map.

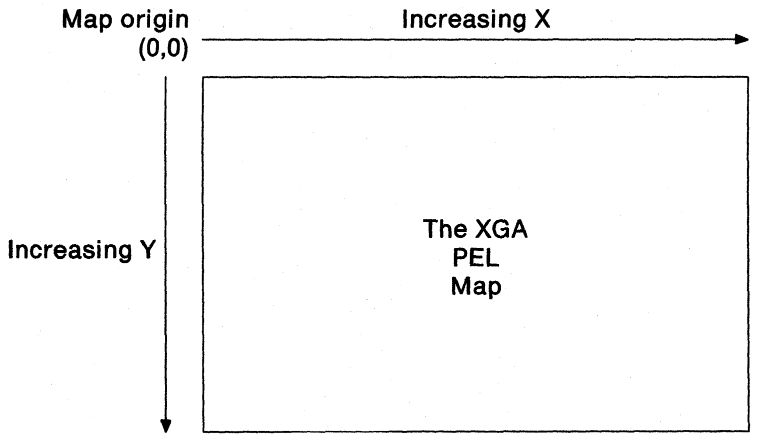


Figure 3-93. XGA PEL Map Origin

In storage, PELs to the right of and below the origin are stored in ascending, contiguous memory locations.

X and Y Pointers

The characteristics of X and Y pointers vary depending on the type of PEL map.

Source and Pattern Maps

These maps each have X and Y pointers that determine the PEL accessed for that map. The two sets of pointers are completely independent, and are modified as the operation proceeds.

If, in the course of an operation, the source or pattern pointers are moved beyond the extremities of the PEL map containing the source or pattern data, they are reset to the opposite edge of the PEL map. Source and pattern maps can be regarded as continuous, as they wrap round at their extremities. This allows a single operation to repeat a small pattern over a large area in the destination map. This is known as pattern tiling, shown in Figure 3-94.

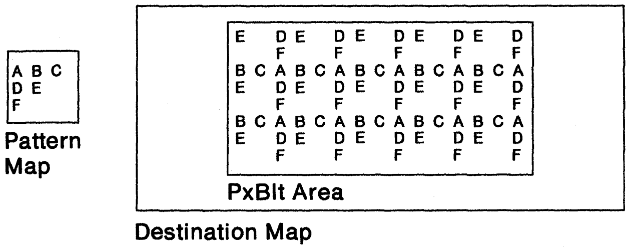


Figure 3-94. Repeating Pattern (Tiling)

Destination Map

If a destination X or Y pointer is moved beyond the extremity of the PEL map containing the destination, the pointers are not wrapped. Updates to the destination are disabled until the pointers are moved to within the defined PEL map. This mechanism is effectively a fixed scissor window around the destination PEL map.

A guardband exists around the destination map to ensure that the destination X and Y pointers do not wrap when they move outside the limits of the map. The guardband is 2048 PELs wide on all sides of the largest definable destination map.

The guardband is illustrated in Figure 3-95.

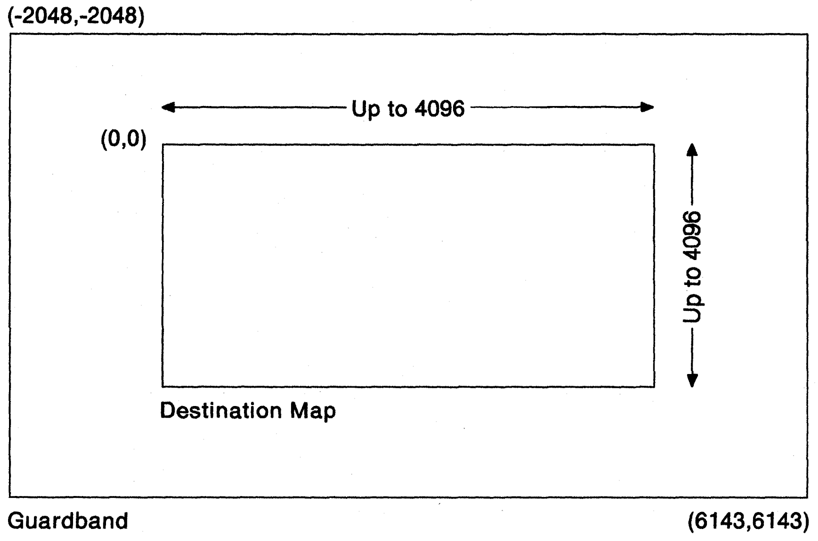


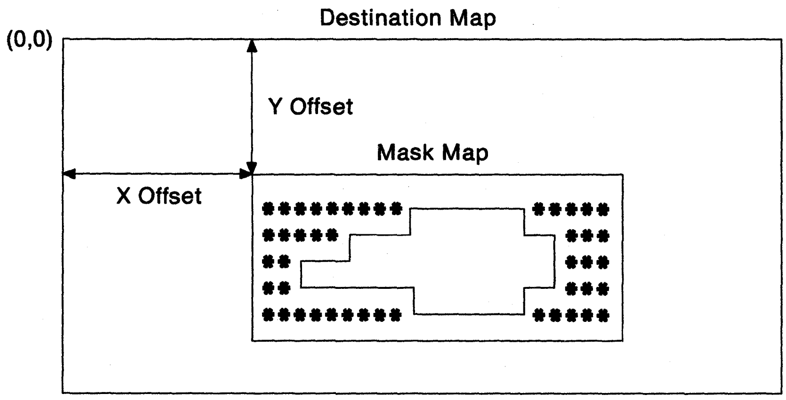
Figure 3-95. Destination Map Guardband

The guardband allows the destination X and Y addresses to range (−2048 to 6143). All PELs within the destination map can be updated, but updates to PELs within the guardband are inhibited. The size of the destination map is determined by the map width and height, so PELs within the range (0,0) to (width − 1, height − 1) can be updated. The guardband occupies PEL X addresses (−2048 to −1), and width to 6143, and Y addresses (−2048 to −1), and height to 6143.

To address the destination map correctly and take advantage of the scissor capability of the coprocessor destination boundary, X and Y destination addresses can be calculated using 16-bit twos complement numbers. All X and Y addresses generated by the operation must be within the range (−2048 to 6143), and all PELs drawn must be inside the bounds of the destination map. Any X and Y addresses generated that are outside the range (−2048 to 6143) cause the X and Y pointers to wrap and produce erroneous results.

Mask Map

The mask map width and height can be any size less than or equal to the dimensions of the destination map. If the mask map is smaller than the destination map, the hardware needs to know where the mask map is positioned relative to the destination map. Two pointers, the mask map origin X offset and mask map origin Y offset, specify the X,Y position where the mask map origin in the destination is located. The following figure illustrates the use of these pointers.



Note: The Mask Map is forced to have the same map origin as the Destination Map.

Figure 3-96. Mask Map Origin X and Y Offsets

The mask map takes its X and Y pointers from the destination map X and Y pointers. For every PEL in the destination map, the corresponding PEL in the mask map is read and, depending on the value of the mask PEL, update of the destination enabled or disabled.

Scissoring with the Mask Map

Hardware scissoring is provided in the coprocessor using the mask map. The mask map can be used for any operation in three ways, as follows:

Disabled
Contents of the mask map and boundary position are ignored.

Boundary Enabled
Contents of the mask map are disabled, but the boundary of the mask map acts as a rectangular scissor window on the destination map. No memory is required to store the map contents in this mode.

Enabled
Contents of the mask map can be used to provide a nonrectangular scissor window. The boundary of the mask map also provides a rectangular scissor window at the extremities of the mask map.

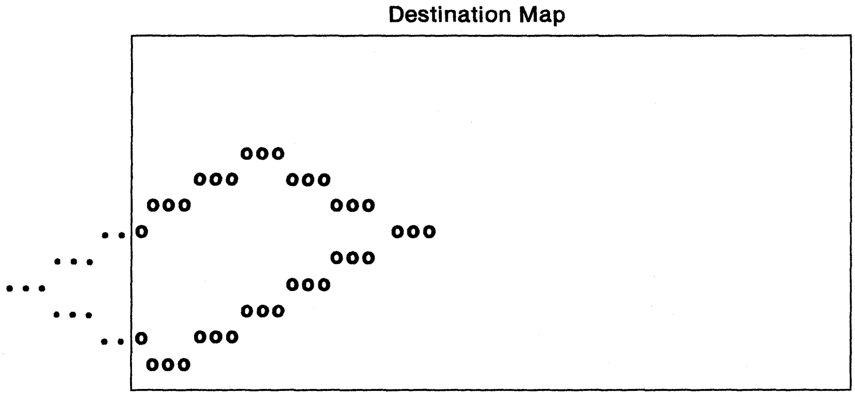
The mask map mode is controlled by a bit in the PEL Operation register. PELs located on a scissor boundary are treated as if they are inside it. The modes are described in the following text.

Mask Map Disabled

When the mask map is disabled, updates to the destination are performed regardless of the position or contents of the mask map. No memory must be reserved for the mask map. The contents of the PEL map M Base Pointer, Width, Height, and Format registers are ignored.

If the current operation attempts to draw outside the boundary of the destination map, the update is automatically inhibited. The destination X and Y pointers are incremented as normal, but destination update is not enabled until the pointers move back inside the bounds of the destination map. A fixed hardware scissor window then exists around the boundary of the destination map. This destination boundary scissor is enabled regardless of the mask map mode.

Figure 3-97 illustrates the destination boundary scissor operation when the mask map is disabled.



o Indicates a PEL drawn
• Indicates a scissored (not drawn) PEL

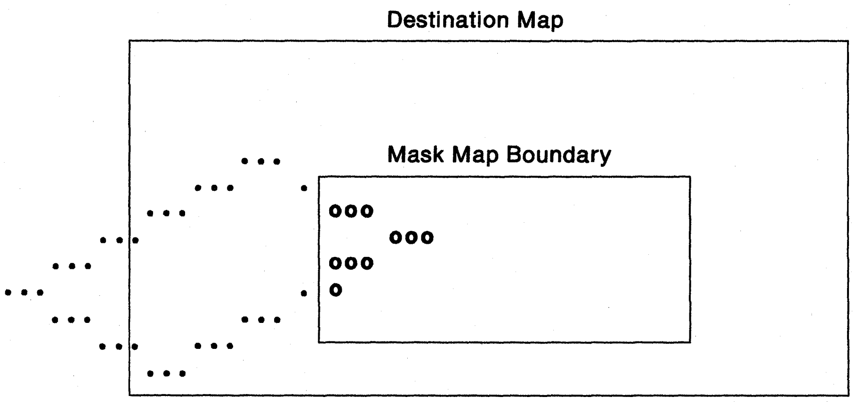
Figure 3-97. Destination Boundary Scissor

Mask Map Boundary Enabled

Mask map boundary enabled mode provides a single rectangular scissor window within the destination map. The contents of the mask map are ignored, so no memory must be reserved for the mask map.

In boundary enabled mode, the size and position of the mask map must be specified. The PEL map M Base Pointer, Width, Height, and Format registers must be defined. These four registers define a rectangular boundary within the destination map. Updates to the destination map inside this boundary take place as normal. Updates outside this boundary are inhibited.

The following figure illustrates a mask map boundary enabled scissoring operation.



o Indicates a PEL drawn
• Indicates a scissored (not drawn) PEL

Figure 3-98. Mask Map Boundary Scissor

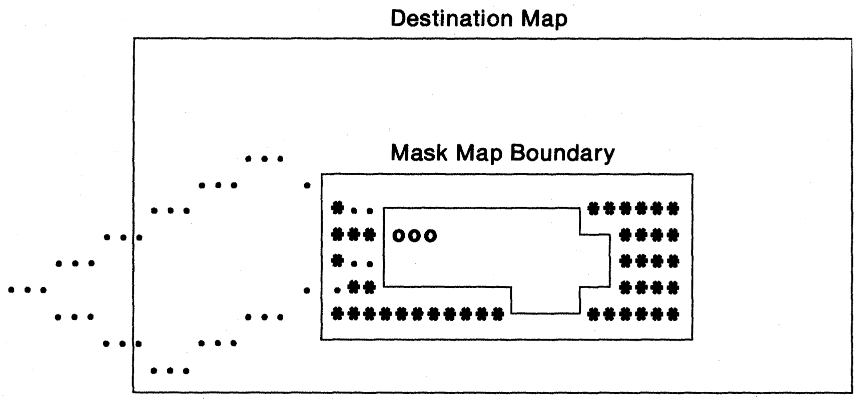
Mask Map Enabled

When the mask map is enabled, both the mask map boundary and contents provide scissoring action. Memory must be reserved to hold the mask map PELs. The PEL map M Base Pointer, Width, Height, and Format registers must be set up to point to the mask data, describing its size and position relative to the destination map.

Any PEL in the destination that is about to be updated has its corresponding mask map PEL examined. If the mask PEL is inactive (0), the destination PEL update is inhibited. If the mask PEL is active (1), the destination PEL is updated as normal. This mode allows drawing nonrectangular scissor windows in the mask map prior to an operation, then in a single execution of an operation, applying a nonrectangular scissor window to that operation.

Memory must be reserved to hold the mask map contents. The mask data is fixed at 1 bit-per-PEL. For a full screen mask map on a 1024 x 768 PEL screen, 96KB of memory are required. If the scissor operation does not cover the whole destination map, a mask map smaller than the destination map can be used to save memory. Applications with no memory available for the mask map contents must use the mask map boundary enabled mode.

The following figure illustrates a mask map enabled scissor operation.



o Indicates a PEL drawn
• Indicates a scissored (not drawn) PEL
✱ Indicates a '0' in the Mask Map

Figure 3-99. Mask Map Enabled Scissor

Before performing an operation that requires a nonrectangular scissor, the nonrectangular mask into the mask map must be drawn. Windowing systems only permit rectangular windows, so the mask can be drawn using a sequence of PxBlt operations that have fixed source data. For more complex shapes, the line draw and draw and step functions can be used to draw area outlines that can then be filled.

A large number of operations can be performed, all using the same mask, keeping the overhead per operation in setting up the mask small. The use of the mask to perform nonrectangular scissors improves the performance of a given drawing operation over a single rectangular scissor that is provided by the hardware.

Drawing Operations

The coprocessor provides four drawing operations:

- Draw and step
- Line draw
- PEL block transfer (PxBlt)
- Area fill.

The operations can be either one-dimensional or two-dimensional. Draw and step and line draw are one-dimensional while the PxBlts are two-dimensional. Draw and step and line draw are collectively called draw operations in the following text.

Either of the draw operations can be read or write. Qualifiers to the operation are described in "Line Draw" on page 3-99.

Draw and Step

This operation draws a PEL at the destination, then updates the X,Y pointers to one of the eight neighbors of the PEL according to a 3-bit code.

Up to 15 address steps can be specified in a fixed direction by each draw and step code. An 8-bit code describes the vector, as shown in Figure 3-100.

7	6	5	4	3	2	1	0
Direction Code			M/D	Number of Steps			

Figure 3-100. Draw and Step Code

Number of Steps

This field indicates how many steps are taken, from 0 to 15. The X,Y pointers are updated after the PEL is drawn, so a draw and step function always attempts to draw at least one PEL.

The number of steps taken in the draw and step operation is one less than the number of PELs that the hardware attempts to draw. When the number of steps is programmed to five, six PELs are drawn; when zero steps are specified, one PEL is drawn. After the draw and step operation, the X,Y pointers point to the last PEL that the operation attempted to draw (this PEL may not be drawn if the last PEL null drawing mode is active).

For example, a draw and step code of hex 35 moves X,Y pointers starting at coordinates (17,10) to coordinates (22,5), as shown in Figure 3-101.

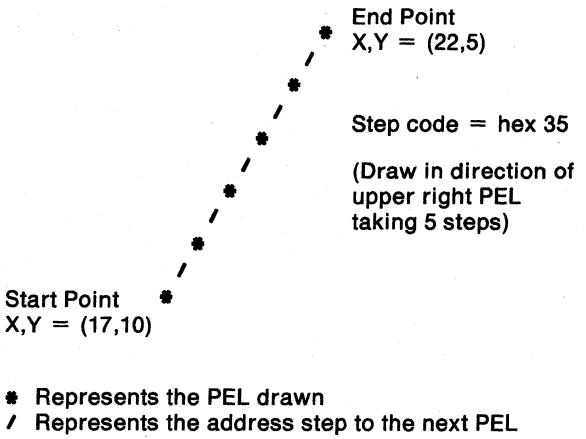

End Point
X,Y = (22,5)
Step code = hex 35
(Draw in direction of upper right PEL taking 5 steps)
Start Point
X,Y = (17,10)
* Represents the PEL drawn
/ Represents the address step to the next PEL

Figure 3-101. Draw and Step Example

M/D

This field specifies if the current operation is a move operation or a draw operation. When set to 1, PELs are drawn. When set to 0, X and Y pointers are modified as normal, but no PELs are drawn.

Direction Code

This field indicates the direction of drawing relative to the current PEL, as shown in Figure 3-102.

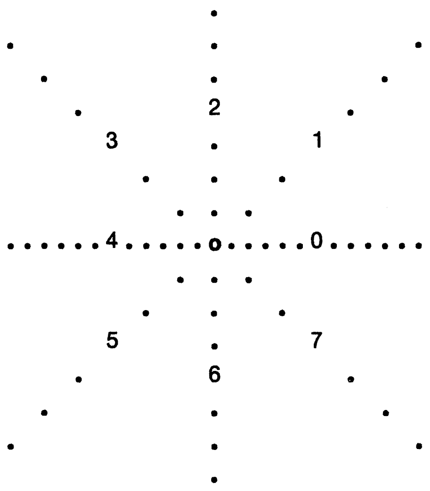


Figure 3-102. Draw and Step Direction Codes

Draw and step codes must be written to the Direction Step register. Each write to the register can load up to four draw and step codes in one access. The draw and step codes are executed starting with the least significant byte. Each group of up to four codes written to the Direction Step register is treated as one operation. All codes are executed before the coprocessor indicates that the operation is complete. However, for the purposes of first and last PEL null drawing, each code describes a distinct line.

The draw and step operation differs from other operations because it is not initiated through the PEL Operations register. Writing a draw and step code to the most significant byte of the Direction Step register initiates the draw and step operation.

Before any data is written to the Direction Step register, the PEL Operation register must be loaded to specify the particular draw and step function and the data flow for the operation. Writing the PEL Operation register with a function of draw and step does not initiate a draw and step operation, but sets up the parameters for the operation. Writing steps to the Direction Step register initiates the draw and step operation. If the PEL Operation register specifies a function other than draw and step when the Direction Step register is written, no operation takes place.

The XGA treats a draw and step code of 00 as a stop code. If a stop code is encountered as one of the four codes in the Direction Step register, the draw and step operation completes after that code has been executed. The completion of the operation is indicated in the normal way through the Coprocessor Control register. The coprocessor busy bit in the Coprocessor Control register indicates the operation has completed because a stop code was encountered. This mechanism allows software to load sequences of draw and step codes to the coprocessor without monitoring the number of codes that make up the figure being drawn.

There are two ways to program fewer than four codes to the Direction Step register. The first unwanted step code can be set to 00 (stop) and all 32 bits of the register written, or only the required number of codes can be written to the Direction Step register. In the latter case, the codes must be written to the most significant bytes of the register. The two methods are shown in Figure 3-103.

Writing 32 bits and using the stop code:

31			0
Don't care	Stop code 00	Code 2	Code 1

Writing only those codes required:

31			0
Code 2	Code 1	Not written	Not written

Note: The figure shows the case when only two Step codes are required. The second method requires the I/O address programmed to change depending on the number of steps written.

Figure 3-103. Programming Fewer Than Four Step Codes

Line Draw

The line draw function uses the Bresenham line drawing algorithm to draw a line of PELs into the destination. The Bresenham line drawing algorithm operates with all parameters normalized to the first octant (octant 0). The octant code for the octant in which the line lies must be specified in the Octant field of the PEL Operation register. This contains a 3-bit code made up of three 1-bit flags called DX, DY, and DZ.

DX is 1 for negative X direction, 0 for positive X direction
DY is 1 for negative Y direction, 0 for positive Y direction
DZ is 1 for $|X| \leq |Y|$, 0 for $|X| > |Y|$ ($|X|$ is the magnitude of X, the value ignoring the sign)

The Octant field is formed by concatenating DX, DY, and DZ. See Figure 3-158 on page 3-149 for octant bit value assignments.

Figure 3-104 shows the encoding of octants.

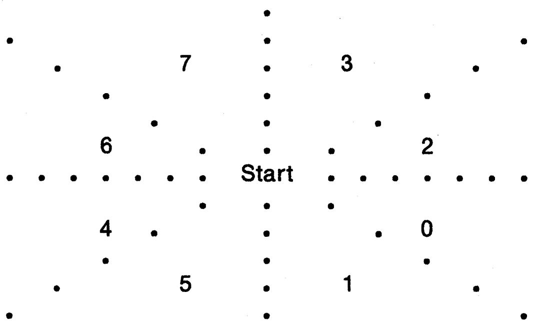


Figure 3-104. Bresenham Line Draw Octant Encoding

The length of the line (delta X when normalized) must be specified in the Operation Dimension 1 register.

The coprocessor provides the following registers to control the draw line address stepping:

- Bresenham Error Term E = 2 × deltaY − deltaX
- Bresenham Constant K1 = 2 × deltaY
- Bresenham Constant K2 = 2 × (deltaY − deltaX).

When the drawing operation has completed, X and Y pointers point at the last PEL of the line.

The coprocessor draw operations that take source data from a PEL map apply the specified address update to either the source or destination map. The X,Y address in the other map is always incremented in X only. There are two possible draw operations, Read Draw and Write Draw.

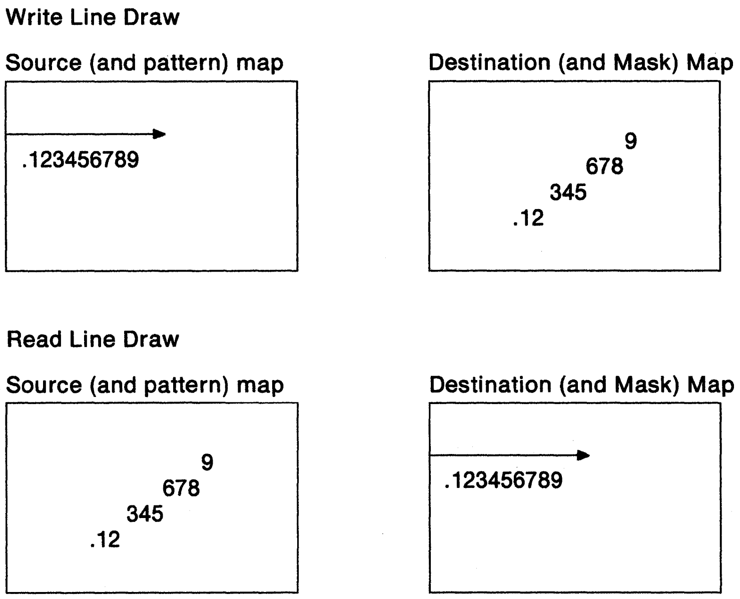
Write Draw After every PEL drawn, the source X,Y pointers are incremented in X only. The destination X,Y pointers are updated according to the current function specified (Bresenham line draw or draw and step).

Read Draw After every PEL drawn, the source X,Y pointers are updated according to the current function specified (Bresenham line draw or draw and step). The destination X,Y pointers are incremented in X only.

The read and write in the terms read draw and write draw refer to the direction of data transfer of the map having its addresses updated by the specified function. During a read line draw, the map where data is read (the source) has its addresses updated by the Bresenham line draw function. During a write draw and step, the map where data is written (the destination) has its addresses updated by the draw and step function.

Note: To draw a fixed color line (by taking the source from the Foreground Color or Background Color register), a write draw function must be used.

Figure 3-105 on page 3-101 illustrates the stepping of X and Y pointers during a read line draw and write line draw.



The numbers 1 to 9 denote each PEL in order of drawing.

Figure 3-105. Memory to Memory Line Draw Address Stepping

In the map that is not having the current addressing function applied, the X pointer is always incremented regardless of the direction of X in the current addressing function. The Y pointer for the same map is not updated during the operation.

The above description refers to the source and destination maps. The pattern map X and Y pointers are updated in the same way as the source pointers. The mask map X and Y pointers (that are not directly accessible), are updated in the same manner as the destination pointers.

If an attempt is made to move any of the map pointers outside the bounds of their current map, the rules set out in "X and Y Pointers" on page 3-87 apply: the source and pattern pointers wrap, and the mask and destination scissor. To draw a line with a repeating color scheme and pattern, the source map width and pattern map width must be set to the required run-length of the repeating colors and pattern respectively. The coprocessor automatically draws the repeating run of colors and pattern. Conversely, if a line with a long nonrepeating color scheme or pattern is required, the source and pattern map widths must be set equal to, or greater than, the line length, otherwise wrapping occurs.

Drawing with Null Endpoint PELs: It is common to draw a series of lines, one after the other, with the endpoint of one line being the starting point of the next line. Such composite lines are called polylines. A problem can arise because the common endpoint of the two abutting lines is drawn twice, once as the last PEL of the first line, and once as the first PEL of the second line. If a mix of XOR is active, the common PEL is drawn and removed. Similar problems arise with different mixes.

To avoid drawing the endpoints of polylines twice, the coprocessor provides functions that inhibit the drawing of the end PEL. Depending on the function selected, either the first PEL or the last PEL of individual lines is not drawn (drawn null). The choice of whether to draw first or last PEL null is arbitrary, as long as one or the other is used for the whole figure being drawn. It is usually a convention of the graphics application whether first or last PEL null is used.

First and last PEL null drawing functions are provided for both the Bresenham line draw function and the draw and step function. In all cases, the programming of parameters is the same as for normal line draw and draw and step. Only the contents of the Drawing Mode field in the PEL Operations register are different.

Area Boundary Drawing: The outline of an object is drawn using Bresenham line draw, draw and step functions, or a combination of the two. The outline is created by observing the following rules.

- If a line is drawn from screen top-to-bottom, draw with last PEL null and draw only the last PEL in every horizontal run of PELs.
- If a line is drawn from screen bottom-to-top, draw with first PEL null, and draw only the first PEL in every horizontal run of PELs.
- If a line is horizontal, draw none of the PELs.
- Always draw with a mix of XOR.

The coprocessor implements these drawing rules in hardware. A shape drawn as an area outline must be drawn as a normal line draw or draw and step operation, with the draw area boundary drawing mode selected in the PEL Operation register and a mix of XOR.

Area Outline Scissoring: It is important during area outline drawing to ensure that the correct outline is drawn when the outline intersects the scissor boundary. In particular, when the outline is scissored by a vertical boundary at the left of a map, a PEL is drawn in the outline to activate filling at that boundary.

Using the combination of mask map and fixed destination boundary scissoring available in XGA, area outlines are incorrectly scissored by the mask map, but correctly scissored by destination map boundary scissoring. The correct area can be filled by ensuring that the mask map scissoring is disabled when the outline is drawn and enabled or boundary enabled when the scan/fill part of the area fill is drawn. This results in the correct, scissored figure being drawn. See "Scissoring with the Mask Map" on page 3-90.

PEL Block Transfer (PxBlt)

The PxBlt function transfers a rectangular block of PELs from the source to the destination. The width and height of the rectangle are specified in the Operation Dimension 1 and Operation Dimension 2 registers. The transfer can be programmed to start at any of the four corners of the rectangle, and proceed toward the diagonally opposite corner. The address is stepped in the X direction until the edge of the rectangle is encountered, then X is reset and the Y direction is stepped. This process is repeated until the entire rectangle is transferred.

PxBlts can be implemented in normal write mode or in read/modify/write mode, depending on the number of bits-per-PEL and the mix being used.

If the PxBlt is being implemented in read/modify/write mode (that is, 1, 2, or 4 bits-per-PEL with *any* mix or 8 bits-per-PEL with a read/modify/write mix), then do one of the following:

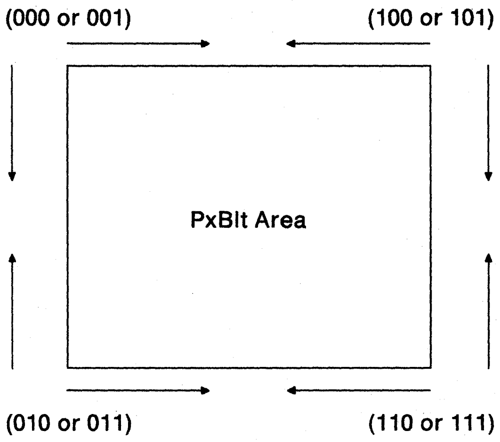
- Ensure that the destination map has a base address that is on a doubleword (4 byte) address boundary, and is an exact number of doublewords wide.
- If the destination map is not doubleword aligned, ensure that the destination map boundary is not crossed during the PxBlt operation.

PxBlt Direction: The PxBlt direction indicates in which direction the X,Y address is stepped across the rectangle. It also defines the starting corner of the transfer. This is significant if the destination rectangle overlaps the source rectangle, so the PxBlt direction must be programmed correctly to achieve the required result.

The direction octant bits in the PEL Operations register determine the direction that the PxBlt is drawn in.

The encoding is as follows:

binary 000 or 001	Start at top left corner of area, increasing right and down.
binary 100 or 101	Start at top right corner of area, increasing left and down.
binary 010 or 011	Start at bottom left corner of area, increasing right and up.
binary 110 or 111	Start at bottom right corner of area, increasing left and up.



Note: Numbers are binary.

Figure 3-106. PxBlt Direction Codes

After a PxBlt operation has completed, the X and Y pointers are set so the X pointer contains its original value at the start of the PxBlt and the Y pointer points to its value on the last line of the PxBlt plus or minus 1, depending on the Y direction that the PxBlt was programmed.

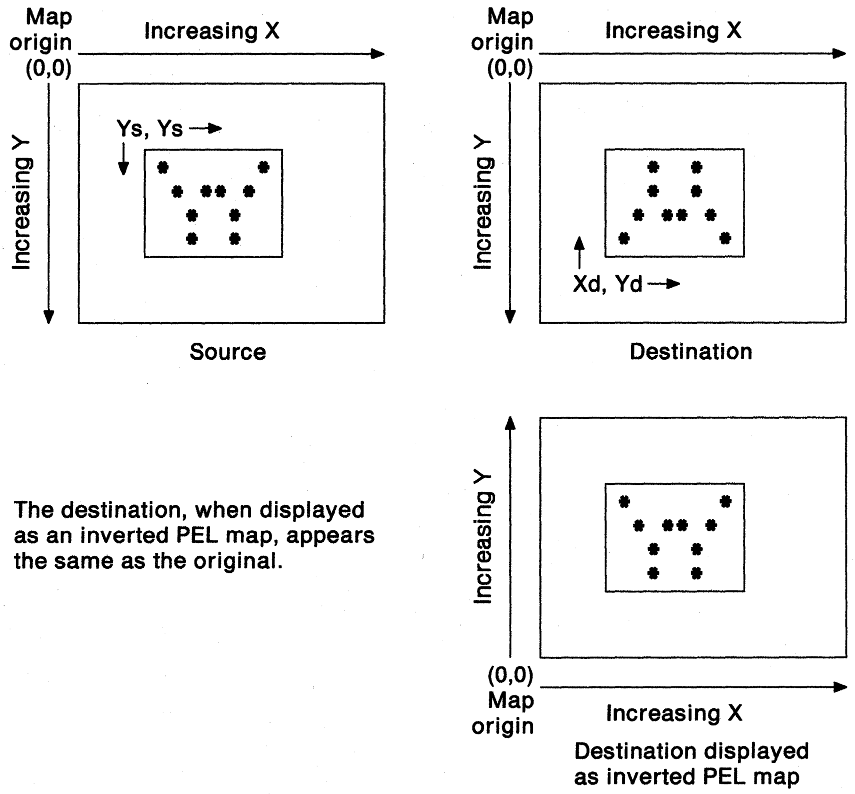
See "Overlapping PxBlts" on page 3-220 for details on PxBlts where the source and destinations overlap.

Inverting PxBlt: As detailed in "Map Origin" on page 3-86, the coprocessor assumes that the origin of a PEL map is at the top left corner of the map, with Y increasing downward. Applications that use an origin at the bottom left of the map (Y increasing upward) use either of the following:

- Modify all Y coordinates by subtracting the map height from them before passing the modified coordinates to the display hardware.
- Use the coprocessor inverting PxBlt operation.

Inverting PxBlt use requires the application to draw into an off-screen PEL map without any Y coordinate modification, and then use the inverting PxBlt operation to move the data to the destination map.

Figure 3-107 on page 3-106 illustrates the X,Y addressing of the inverting PxBlt operation, and shows how the result of the inverting PxBlt appears the same as the original when displayed as an inverted PEL map (that is, with the origin at the bottom left).



The destination, when displayed as an inverted PEL map, appears the same as the original.

Figure 3-107. Inverting PxBlt

An inverting PxBlt is set up in the same manner as a standard PxBlt with the following notes:

- The PxBlt direction set applies to the updating of the source X and Y addresses.
- The destination Y pointer must be programmed to the opposite (in Y) corner of the destination rectangle.
- The Function field in the PEL Operation register must be set to inverting PxBlt as opposed to PxBlt.

See "Overlapping PxBlts" on page 3-220 for details on PxBlts when the source and destinations overlap.

Area Fill

The following steps are required to perform an area fill operation without a user pattern:

1. Draw the closed outline of the area to be filled using the area boundary drawing mode. Typically, a unique, off-screen PEL map would be defined to draw the area boundary into. This PEL map must be initialized to contain 0-value PELs before the boundary is drawn. This PEL map must be in a 1 bit-per-PEL format.
2. Designate the PEL map where the area boundary was drawn as the pattern map.
3. Specify the desired destination.
4. Select the desired foreground mix and source.
5. Specify the background mix as Destination (code 5).
6. Specify the operation direction as any direction with X increasing (Codes 0 or 1, 2 or 3), because the pattern data is scanned from left to right. Selection of a negative X direction code for area fill operations results in fill errors.
7. Initiate the area fill operation.

During the area fill operation, the coprocessor applies a filling function to the pattern PELs before they are used to select background and foreground sources and mixes in the usual way. The filling function modifies the pattern PELs horizontally line by line. It scans the pattern from left to right, and when encountering the first foreground (1) PEL, sets all subsequent PELs to foreground (1) until the next foreground PEL is encountered.

This process is illustrated in Figure 3-108.

```
Pattern scanned to the right ───►


Original Pattern        0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0
                                              │
                                              │ Filling function
                                              ▼
Filled Pattern          0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0
```

Figure 3-108. Pattern Filling

The filled pattern is generated internally to the coprocessor. It is then used exactly as the pattern in any normal operation, with foreground (1) PELs selecting foreground source and mix, and background (0) PELs selecting background source and mix. During area fill operations, it is required to fill the specified area and leave all other PELs unchanged. This is why the background mix was specified to be destination. Figure 3-92 on page 3-82 shows the position of the pattern-filling circuitry in the coprocessor data flow.

Area fill operations that require a pattern fill must be performed in two stages. This is because area fill PxBlt operations use the pattern map to perform the area fill function and cannot include a user pattern in a single operation. However, by first combining the contents of the mask map with a mask of the filled area, a full pattern PxBlt of an area can be achieved as follows:

1. Both the pattern map and the destination map must be defined as the map containing the previously drawn area boundary. The source map must be defined as the map that would normally supply the mask map for the operation. The mask map facility must be disabled. An area fill PxBlt must be performed with the following conditions:

   - Foreground source = source PEL map
   - Foreground mix = Source (code 3)
   - Background source = background color
   - Background color = 0
   - Background mix = Source (code 3).

   Note: All the maps in this operation must be 1 bit-per-PEL.

   This operation combines the mask data for the pattern area fill with a mask of the filled area.

2. A second non-area fill PxBlt must be performed with the combined mask generated in step 1 defined as the mask map. All other maps can be used as normal with no restrictions.

Logical and Arithmetic Functions

During an operation in the coprocessor, source data is combined with destination data, under the control of pattern data, and the result is written back to the destination. Mask data can be included in the operation to selectively inhibit updating of destination data.

Source data can be either foreground source or background source on a PEL-by-PEL basis. The foreground source is combined with the destination using the foreground mix; the background source is combined with the destination using the background mix. The pattern determines if the source and mix are foreground or background for a particular PEL. If the pattern PEL is 1, source and mix are foreground; if it is 0, they are background.

The foreground and background sources can each be either a fixed color over the whole operation or PEL data taken from the source PEL map. The background source and foreground source bits in the PEL Operations register determine whether fixed colors or source PEL map data is used in an operation. The fixed color that is used as the foreground source is called the foreground color, and is stored in the Foreground Color register. The fixed color that is used as the background source is called the background color, and is stored in the Background Color register.

The possible combinations of source, destination and pattern are shown below:

- Pattern PEL = 1 (foreground source)
  - New destination PEL = old destination PEL **Fgd OP** Foreground color
  - New destination PEL = old destination PEL **Fgd OP** PEL map source.
- Pattern PEL = 0 (background source)
  - New destination PEL = old destination PEL **Bgd OP** Background color
  - New destination PEL = old destination PEL **Bgd OP** PEL map source.

Fgd OP is the logical or arithmetic function specified in the Foreground Mix register. **Bgd OP** is the logical or arithmetic function specified in the Background Mix register.

These operations can be inhibited by the contents of the mask map. If the mask PEL is 0, the destination PEL is not modified. If the mask PEL is 1, the selected operation is applied to the destination PEL.

Mixes

The foreground and background mixes provided by the XGA are independent. The XGA provides all logical mixes of two operands and six arithmetic mixes. The mixes provided are as follows:

Code (hex)	Function
00	Zeros
01	Source AND Destination
02	Source AND NOT Destination
03	Source
04	NOT Source AND Destination
05	Destination
06	Source XOR Destination
07	Source OR Destination
08	NOT Source AND NOT Destination
09	Source XOR NOT Destination
0A	NOT Destination
0B	Source OR NOT Destination
0C	NOT Source
0D	NOT Source OR Destination
0E	NOT Source OR NOT Destination
0F	Ones
10	Maximum
11	Minimum
12	Add With Saturate
13	Subtract (Destination – Source) With Saturate
14	Subtract (Source – Destination) With Saturate
15	Average

Note: Mix codes hex 16 to FF are reserved.

Figure 3-109. Foreground and Background Mixes

Saturate means that if the result of an arithmetic operation is greater than all 1's, the final result remains all 1's. If the result of an arithmetic operation is less than 0, the final result remains at 0.

Breaking the Coprocessor Carry Chain

To limit the operation of the coprocessor to certain bits in a PEL (for example, to perform an operation on both the upper and lower 4 bits of an 8-bit PEL independently), it is not desirable for the arithmetic operations to propagate a carry from one group of bits in the PEL to the next. One solution is to use the XGA PEL bit mask to ensure only one component of the PEL is processed at a time. The disadvantage of this technique is that the operation must be repeated once for each component in the PEL.

The XGA provides an alternative mechanism that allows PELs with component fields to process correctly in one pass. A carry chain mask can be specified that determines how carry bits are propagated in the coprocessor. By loading the appropriate mask in the Carry Chain Mask register before performing an operation involving an arithmetic operation or color compare, the PEL is effectively divided into independent fields. The mask prevents the coprocessor carry being propagated across the field boundaries.

Each bit in the mask enables or disables the propagation of the carry from the corresponding bit in the coprocessor to its more significant neighbor. The mask is n − 1 bits wide for a PEL n bits wide, and the carry from the most significant bit of the coprocessor is not propagated.

An example for a carry chain mask for an 8-bit PEL with two 4-bit fields follows:

7	6	5	4	3	2	1	0
-	1	1	1	0	1	1	1

Figure 3-110. Carry Chain Mask for an 8-bit PEL

Bits outside the required mask size for a given PEL size need not be written in the register.

Generating the Pattern from the Source

Pattern data for an operation can be supplied by PEL maps A, B, or C, or it can be fixed to 1 (foreground source) throughout the operation. Pattern data can also be internally generated by the coprocessor from source PEL map data. A comparison operation is performed on each source PEL and the pattern data is generated depending on the result.

The comparison operation compares the source PEL to 0. For any source PEL with a value of 0, a 0 (background) pattern PEL is generated. For any nonzero source PEL, a 1 (foreground) pattern PEL is generated. The internally generated pattern is then used to select between foreground and background sources, and mixes in the usual way. When the pattern is internally generated, the coprocessor ignores the pattern PEL map contents.

This capability allows the background source data and mix to be forced for all 0 value PELs in the source. In particular, it provides a transparency function, where a multibit character can be drawn onto a destination with the destination data showing through any 0 (black) PEL in the source character definition.

Color Expansion

If the source PELs for an operation have fewer bits per PEL than the destination PELs, the source PELs must be expanded to the same size as those in the destination before they are combined. This process is referred to as color expansion.

The major use of color expansion is to draw 1-bit-per-PEL character sets on n-bits-per-PEL destinations. The coprocessor performs this function in hardware, but does not have a color expansion look-up table. Instead, the 1-bit-per-PEL character map must be defined as the pattern map. The PEL Operation register must be programmed to use the Foreground Color and Background Color registers, not the source map. The Foreground Color and Background Color registers then act as a two-entry color expansion look-up table, and the character map is expanded to the number of bits per PEL in the destination.

PEL Bit Masking

The PEL bit mask allows any combination of bits in a destination PEL to be protected from update (being written). A mask value must be loaded in the PEL Bit Mask register to enable or disable updating of PEL bits selectively as required.

This mask is the same as the plane mask in subsystems that are plane oriented, as opposed to packed-PEL.

When the destination bits-per-PEL is less than 8 bits, only the low order bits of the PEL Bit Mask register are significant.

A bit that is not write enabled is prevented from affecting arithmetic or compare operations. In effect, masked bits are completely excluded from the operation or comparison.

Color Compare

The value that the destination PELs are compared with is stored in the Destination Color Compare Value register. The Destination Color Compare Condition register indicates the condition when the destination update is inhibited. The possible conditions are as follows:

Condition Code	**Condition**
0	Always True (disable update)
1	Destination Data > Color Compare Value
2	Destination Data = Color Compare Value
3	Destination Data < Color Compare Value
4	Always False (enable update)
5	Destination Data > = Color Compare Value
6	Destination Data < > Color Compare Value
7	Destination Data < = Color Compare Value

Figure 3-111. Color Compare Conditions

Note: A comparison result of true prevents update to the destination.

Controlling Coprocessor Operations

Starting a Coprocessor Operation

Coprocessor operations are started by writing the most significant byte of the PEL Operations register. One exception to this is the draw and step function. For details, see "Draw and Step" on page 3-95.

Suspending a Coprocessor Operation

Coprocessor operations can be suspended before they have completed. The state of the coprocessor, including internal register contents, can then be read rapidly to allow task state saving. A previous task can be restored through the same data port and the restored operation can be restarted.

The suspend operation bits in the Coprocessor Control register are used to suspend and restart coprocessor operations.

If a coprocessor operation is suspended, a terminate operation is required before starting a new coprocessor operation.

Terminating a Coprocessor Operation

Operations can be terminated before they have completed. The state of the coprocessor registers that are updated as the operation proceeds is undefined after the operation is terminated and their contents must not be relied upon. "Coprocessor Registers" on page 3-118 details the registers that are updated as an operation proceeds.

The terminate operation bit in the Coprocessor Control register is used to terminate operations.

Coprocessor Operation Completion

There are two methods for the system microprocessor to detect the completion of a coprocessor operation:

- Receive an operation-complete interrupt from the XGA.
- Poll the coprocessor busy bit in the Coprocessor Control register.

Coprocessor-Operation-Complete Interrupt

The coprocessor provides an operation-complete interrupt that can interrupt the system on completion of an operation. The interrupt is enabled by a bit in the Interrupt Enable register and its status is indicated by a bit in the Interrupt Status register. See "Interrupt Enable Register (Address 21x4)" on page 3-34 and "Interrupt Status Register (Address 21x5)" on page 3-36 for bit locations.

Regardless of the state of the operation-complete interrupt enable bit, the status bit is always set to 1 on completion of an operation. The application must ensure that this bit is reset before starting an operation. This is done by writing a 1 back to the status bit.

If the interrupt enable bit is 1, the completion of an operation not only sets the interrupt status bit, but also causes an interrupt. The system microprocessor must reset the interrupt by writing a 1 back to the status bit after servicing the interrupt.

Coprocessor Busy Bit

The coprocessor busy bit in the Coprocessor Control register indicates if the coprocessor is executing an operation. It is set to 1 by the hardware when the coprocessor is executing an operation and reset to 0 when the operation completes. Applications can read this bit to determine if the coprocessor is busy. See "Waiting for Hardware Not Busy" on page 3-221 for more information.

Accesses to the Coprocessor During an Operation

When the coprocessor is executing an operation, the system processor can only perform read accesses to the coprocessor registers. Write accesses are not permitted because they could corrupt operation data.

If the system processor attempts to write data to the coprocessor registers during an operation, the coprocessor allows the access to complete, but the executing operation may be corrupted. The coprocessor interrupts the system microprocessor to indicate a write access occurred during an active operation and the operation may have been corrupted. This interrupt is called the coprocessor-access-rejected interrupt. An enable bit is in the Interrupt Enable register and a status bit is in the Interrupt Status register. See "Interrupt Enable Register (Address 21x4)" on page 3-34 and "Interrupt Status Register (Address 21x5)" on page 3-36 for bit locations.

There is one exception to this rule. The Coprocessor Control register can be written during an operation without corrupting the operation. See "Coprocessor Control Register (Offset 11)" on page 3-122 for more information.

Coprocessor State Save/Restore

When operating in a multitasking environment it is necessary to save and restore the state of the display hardware when switching tasks.

Sometimes a task switch is required when the coprocessor is in the course of executing an operation. Not only the contents of registers visible to the system microprocessor, but also contents of internal registers (the state of the coprocessor) must be saved and, later, restored. The coprocessor has special hardware that lets it suspend the execution of an operation and efficiently save and restore task states.

Suspending Coprocessor Operations

At any time during the execution of a coprocessor operation, the operation can be suspended by writing to a bit in the Coprocessor Control register. Any executing memory cycle is completed before the coprocessor suspends the operation. The system can then save and restore the coprocessor contents and restart the restored operation by clearing the bit in the Coprocessor Control register.

If a coprocessor operation is suspended, a terminate operation is required before starting a new coprocessor operation.

Save/Restore Mechanism

The coprocessor provides two special 32-bit save/restore data ports. All the coprocessor state data passes through these ports when the state is being saved or restored. The number of doublewords read or written is determined by two read-only registers (State Length registers A and B). The amount of data saved or restored is less than 1KB. State-saving software must perform string I/O read instructions, reading data from the two save/restore data ports in turn. The coprocessor hardware automatically provides successive doublewords of data on successive reads. After the state has been saved, the coprocessor is in a reset state.

If a coprocessor operation is suspended, a terminate operation is required before starting a new coprocessor operation.

Restoring the state of the coprocessor uses a similar process. State data must be moved back into the coprocessor using string I/O write instructions. The state data must be written back into the coprocessor in the same order as it was read (first out, first in).

The exact number of doublewords specified in the State Length registers must be read or written when saving or restoring the coprocessor state. Failure to do this leaves the coprocessor in an indeterminate state.

Coprocessor Registers

The XGA coprocessor supports two register interface formats. The type of interface required (Intel or Motorola) is set when selecting Extended Graphics mode in the Operating Mode register.

The difference between Intel and Motorola formats is that, with two exceptions, the bytes within each 4 bytes of register space are reversed (byte 0 becomes byte 3). The two exceptions are the Direction Steps register and the PEL Operations register. The bytes within these registers are not reversed because the existing byte order is required by the operation being performed.

Most of the coprocessor registers are not directly readable by the system microprocessor. Registers that cannot be read directly can be read indirectly using the coprocessor state save and restore mechanism. See "Save/Restore Mechanism" on page 3-117 for more information. Only the following registers are readable directly by the system microprocessor:

- State Save/Restore Data Ports register
- State Length registers
- Coprocessor Control register
- Virtual Memory Control register
- Virtual Memory Interrupt Status register
- Current Virtual Address register
- Bresenham Error Term E register
- Source X Address and Source Y Address registers
- Pattern X Address and Pattern Y Address registers
- Destination X Address and Destination Y Address registers.

The contents of most coprocessor registers are not changed during a coprocessor operation and therefore do not need to have their contents reloaded before starting another similar operation. The registers with contents that change during an operation are:

Bresenham Error Term E
The error term is updated throughout line draw operations.

Source X Address and Source Y Address
Any operation that uses the source PEL map updates these pointers.

Pattern X Address and Pattern Y Address
The pattern map X and Y pointers are updated during any operation that does not have the Pattern field in the PEL Operation register set to foreground.

Destination X Address and Destination Y Address
The destination map X and Y pointers are updated during all operations.

The following figures show the coprocessor register space in Intel and Motorola formats.

Coprocessor Address Space				
Byte 3	Byte 2	Byte 1	Byte 0	
Page Directory Base Address				0
Current Virtual Address				4
				8
		State B length	State A length	C
	PEL Map Index	Coprocessor Control		10
PEL Map n Base Pointer				14
PEL Map n Height		PEL Map n Width		18
			PEL Map n Format	1C
		Bresenham Error Term		20
		Bresenham K1		24
		Bresenham K2		28
Direction Steps				2C
				30
				34
				38
				3C
				40
				44
	Destination Color Compare Condition	Background Mix	Foreground Mix	48
Destination Color Compare Value				4C
PEL Bit Mask				50
Carry Chain Mask				54
Foreground Color Register				58
Background Color Register				5C
Operation Dimension 2		Operation Dimension 1		60
				64
				68
Mask Map Origin Y Offset		Mask Map Origin X Offset		6C
Source Map Y Address		Source Map X Address		70
Pattern Map Y Address		Pattern Map X Address		74
Destination Map Y Address		Destination Map X Address		78
PEL Operation				7C

Figure 3-112. XGA Coprocessor Register Space, Intel Format

Note: All unused and undefined offsets are reserved.

Coprocessor Address Space				
Byte 0	Byte 1	Byte 2	Byte 3	
Page Directory Base Address				0
Current Virtual Address				4
				8
		State B length	State A length	C
	PEL Map Index	Coprocessor Control		10
PEL Map n Base Pointer				14
PEL Map n Height		PEL Map n Width		18
			PEL Map n Format	1C
		Bresenham Error Term		20
		Bresenham K1		24
		Bresenham K2		28
Direction Steps				2C
				30
				34
				38
				3C
				40
				44
	Destination Color Compare Condition	Background Mix	Foreground Mix	48
Destination Color Compare Value				4C
PEL Bit Mask				50
Carry Chain Mask				54
Foreground Color Register				58
Background Color Register				5C
Operation Dimension 2		Operation Dimension 1		60
				64
				68
Mask Map Origin Y Offset		Mask Map Origin X Offset		6C
Source Map Y Address		Source Map X Address		70
Pattern Map Y Address		Pattern Map X Address		74
Destination Map Y Address		Destination Map X Address		78
PEL Operation				7C

Figure 3-113. XGA Coprocessor Register Space, Motorola Format

Note: All unused and undefined offsets are reserved.

Register Usage Guidelines

Unless otherwise stated, the following are guidelines to be used when accessing the coprocessor registers:

- Special reserved register bits must be used as follows:
  - Register bits marked with '–' must be set to 0. These bits are undefined when read and should be masked off if the contents of the register is to be tested.
  - Register bits marked with '#' are reserved and the state of these bits must be preserved. When writing the register, read the register first and change only the bits that must be changed.
- Unspecified registers or registers marked as reserved in the XGA coprocessor address space are reserved. They must not be written to or read from.
- During a read, the values returned from write-only registers are reserved and unspecified.
- The contents of read-only registers must not be modified.
- Counters must not be relied upon to wrap from the high value to the low value.
- Register fields defined with valid ranges must not be loaded with a value outside the specified range.
- Register field values defined as reserved must not be written.

The following sections describe the coprocessor registers in detail. Unless stated otherwise, the register definitions are in Intel format.

Virtual Memory Registers

The XGA coprocessor virtual memory implementation is given in "Virtual Memory Description" on page 3-156.

State Save/Restore Registers

The following registers allow the internal state of the coprocessor to be saved and restored. “Coprocessor State Save/Restore” on page 3-116 describes this mechanism.

Coprocessor Control Register (Offset 11)

This read/write register has an offset of hex 11. The Coprocessor Control register indicates if the coprocessor is currently executing an operation. The current coprocessor operation can be terminated or suspended by writing to this register.

7	6	5	4	3	2	1	0	
–	–	TO	–	SO	–	SR	–	(Write Format)

7	6	5	4	3	2	1	0	
BSY	–	TO	OS	SO	–	SR	–	(Read Format)

–	:	Set to 0, Undefined on Read
BSY	:	Coprocessor Busy
TO	:	Terminate Operation
OS	:	Operation Suspended
SO	:	Suspend Operation
SR	:	State Save/Restore

Figure 3-114. Coprocessor Control Register, Offset Hex 11

The register fields are defined as follows:

BSY When the Coprocessor Busy field (bit 7) is read as 1, the coprocessor is currently executing an operation. When read as 0, the coprocessor is idle.

TO	Coprocessor operations can be terminated by writing a 1 to the Terminate Operation field (bit 5). The application must ensure that the operation has terminated before proceeding. Termination is ensured by waiting for the operation-complete interrupt (if enabled), or by reading the Coprocessor Busy field until the coprocessor goes not busy (bit 7 = 0). After the coprocessor has terminated the operation, it automatically sets the Terminate Operation field to 0. The coprocessor is returned to its initial power-on state, with coprocessor interrupts masked off and certain other register bits reset. All registers must be assumed invalid and must be reprogrammed before another operation is initiated.
OS	The Operation Suspended field (bit 4) is set to 1 by the coprocessor when it has suspended the operation. This bit must be read by the system microprocessor to ensure that an operation has been suspended before saving/restoring is started.
SO	When the Suspend Operations field (bit 3) is set to 1, coprocessor operations are suspended. When set to 0, the suspended processor operations are restarted. This must be done to restart a restored operation after a task switch. When the operation restarts, the coprocessor resets the Operations Suspended field to 0. Suspending an operation flushes the translate look-aside buffer. If a coprocessor operation is suspended, a terminate operation is required before starting a new coprocessor operation.
SR	The State Save/Restore field (bit 1) selects whether to save or restore the coprocessor state. When set to 0, a state restore can be performed; when set to 1, a state save can be performed. The Coprocessor Control register must be written with the Suspend Operation field set and the Save/Restore field appropriately set before each state save or state restore.

State Length Registers (Offset C and D)

These read-only registers have offsets of hex C and D. *Do not write to* these registers. The State Length registers return the length, in doublewords, of parts A and B of the coprocessor state for save and restore.

Save/Restore Data Ports Register (I/O Index C and D)

These read/write registers have I/O indexes of hex C and D. The Save/Restore registers are directly mapped to I/O address space and do not appear in the coprocessor register summary. However, they are coprocessor registers and are described here.

These registers are used to save and restore the two parts, A and B, of the internal state of the coprocessor. After a state save or restore is initiated, string I/O reads or writes must be executed from or to these registers. The data can be read or written using any combination of byte, word, or doubleword accesses, provided that the exact number of doublewords specified in the State Length registers is read or written. Failure to read or write the correct amount of data leaves the coprocessor in an undefined state.

Data must be written back to this port in the order it was read (first out, first in).

PEL Interface Registers

The following is a detailed description of the coprocessor PEL interface registers.

PEL Map Index Register (Offset 12)

This write-only register has an offset of hex 12.

7	6	5	4	3	2	1	0
–	–	–	–	–	–	PMI	

```
  - : Set to 0
PMI : PEL Map Index
```

Figure 3-115. PEL Map Index Register, Offset Hex 12

The register field is defined as follows:

PMI The PEL Map Index field (bits 1, 0) selects which PEL map registers will be used.

Each PEL map used in the XGA is described by four registers, as follows:

- The PEL Map n Base Pointer register
- The PEL Map n Width register
- The PEL Map n Height register
- The PEL Map n Format register.

Each PEL map has its own copy of these registers, so there are four copies of these registers in the XGA, one each for:

- The mask map
- PEL map A
- PEL map B
- PEL map C.

Only one of these banks of PEL map registers is visible to the system microprocessor at any time. The PEL Map Index register is used to select the maps the registers apply to. The encoding of the PEL Map Index register is shown in the following figure.

PMI Field (binary)	PEL Map Register
0 0	Mask Map
0 1	PEL Map A
1 0	PEL Map B
1 1	PEL Map C

Figure 3-116. PEL Map Index

Before loading the PEL map base pointer, width, height, and format for a particular map, the PEL Map Index register must be set up to point to the registers of the required map. For example, to set up the register for map B, first load the PEL map index with binary 10.

PEL Map n Base Pointer Register (Offset 14)

This write-only register has an offset of hex 14. The n is selected using the "PEL Map Index Register (Offset 12)" on page 3-125.

```
 31                                                 0
┌────────────────────────────────────────────────────┐
│                 PEL Map n Pointer                  │
└────────────────────────────────────────────────────┘
```

Figure 3-117. PEL Map n Base Pointer Register, Offset Hex 14

The PEL Map n Pointer register (bits 31 – 0) specifies the byte address in memory of the start of a PEL map. If virtual address mode is enabled, this address is a virtual address, otherwise it is a physical address.

PEL Map n Width Register (Offset 18)

This write-only register has an offset of hex 18. The PEL Map n Width register can be loaded with any value in the range (0 to 4095). The n is selected using the “PEL Map Index Register (Offset 12)” on page 3-125.

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
┌────────────────────────────────────────────────┐
│                 PEL Map n Width                │
└────────────────────────────────────────────────┘
```

Figure 3-118. PEL Map n Width Register, Offset Hex 18

The PEL Map n Width register (bits 15 – 0) specifies the width of a PEL map. The width of a PEL map is measured in PELs, that is, independent of the number of bits-per-PEL.

Widths are used during address stepping to specify the width of the PEL map. Steps with a Y direction component are achieved by the hardware adding or subtracting the width ± 0/1.

The PEL map width is also used for wrapping the source and pattern maps, or to implement the fixed scissor boundary around the destination map.

The value loaded in the width register must be 1 less than the bit map width. For a bit map that is 1024 PELs wide, the width register must be loaded with 1023 (hex 03FF).

PEL Map n Height Register (Offset 1A)

This write-only register has an offset of hex 1A. The PEL Map n Height register can be loaded with any value in the range (0 to 4095). The n is selected using the “PEL Map Index Register (Offset 12)” on page 3-125.

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
┌───────────────────────────────────────────────┐
│                PEL Map n Height               │
└───────────────────────────────────────────────┘
```

Figure 3-119. PEL Map n Height Register, Offset Hex 1A

The PEL Map n Height register (bits 15 – 0) specifies the height of a PEL map. The height of the PEL map is measured in PELs, that is, independent of the number of bits-per-PEL.

The PEL map height is used for wrapping the source and pattern maps, or to implement the fixed scissor boundary around the destination map.

The value loaded in the height register must be one less than the PEL map height. For a bit map that is 768 PELs high, the height register must be loaded with 767 (hex 02FF).

PEL Map n Format Register (Offset 1C)

This write-only register has an offset of hex 1C. The n is selected using the "PEL Map Index Register (Offset 12)" on page 3-125.

7	6	5	4	3	2	1	0
–	–	–	–	PO	PS		

```
 - : Set to 0
PO : PEL Order
PS : PEL Size
```

Figure 3-120. PEL Map n Format Register, Offset Hex 1C

The register fields are defined as follows:

PO The PEL Order field (bit 3) selects the format for the memory-to-screen mapping. When set to 0, the PEL map is Intel-ordered; when set to 1, the PEL map is Motorola-ordered. "PEL Formats" on page 3-83 describes the difference in formats.

PS The PEL Size field (bits 2–0) specifies the number of bits-per-PEL in the PEL map. PEL maps occupied by the source or destination map can be 1, 2, 4, or 8 bits-per-PEL. The PEL map occupied by the pattern map must be 1 bit-per-PEL. Programming the pattern to be taken from a PEL map that does not contain 1-bit PELs produces undefined results. The PEL size field definitions are shown in the following figure.

PS Field (binary)	PEL Size
0 0 0	1 Bit
0 0 1	2 Bits
0 1 0	4 Bits
0 1 1	8 Bits
1 0 0	Reserved
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

Figure 3-121. PEL Size Value Assignments

PEL Maps A, B, and C

PEL maps A, B, and C are all described by similar registers. The different maps are merely three instances of PEL maps that can have different locations in memory, sizes, and formats.

The pattern map used by the XGA must be 1 bit-per-PEL. The pattern map must reside in a PEL map that is 1 bit-per-PEL. Failure to do this produces undefined results.

Mask Map

The mask map has a base pointer, width, and height that are similar to those of PEL maps A, B, and C.

The Mask Map Format register differs from maps A, B, and C in that only the Motorola/Intel format bit of the mask map is programmable. This register bit operates the same as the bit for maps A, B, and C. The number of bits-per-PEL is assumed to be 1 bit-per-PEL. The bits-per-PEL must always be set to 1 bit-per-PEL to ensure future compatibility.

Bresenham Error Term E Register (Offset 20)

This read/write register has an offset of hex 20.

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
+----------------------------------------------+
|               Bresenham Error Term           |
+----------------------------------------------+
```

Figure 3-122. Bresenham Error Term E Register, Offset Hex 20

The Bresenham Error Term register (bits 15–0) specifies the Bresenham error term for the draw line function. The error term value is a signed quantity, calculated as ((2 × deltaY) – deltaX) after normalization to the first octant.

This register must be written as a 16-bit sign-extended twos complement number in the range (–8192 to 8191).

Bresenham Constant K1 Register (Offset 24)

This write-only register has an offset of hex 24.

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
┌───────────────────────────────────────────────┐
│               Bresenham Constant K1           │
└───────────────────────────────────────────────┘
```

Figure 3-123. Bresenham Constant K1 Register, Offset Hex 24

The Bresenham Constant K1 register (bits 15 − 0) specifies the Bresenham constant, K1, for the draw line function. The K1 value is a signed quantity, calculated as (2 × deltaY) after normalization to the first octant.

This register must be written as a 16-bit sign-extended twos complement number in the range (−8192 to 8191).

Bresenham Constant K2 Register (Offset 28)

This write-only register has an offset of hex 28.

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
┌───────────────────────────────────────────────┐
│               Bresenham Constant K2           │
└───────────────────────────────────────────────┘
```

Figure 3-124. Bresenham Constant K2 Register, Offset Hex 28

The Bresenham Constant K2 register (bits 15 − 0) specifies the Bresenham constant, K2, for the draw line function. The K2 value is a signed quantity, calculated as (2 × (deltaY − deltaX)) after normalization to the first octant.

This register must be written as a 16-bit sign-extended twos complement number in the range (−8192 to 8191).

Direction Steps Register (Offset 2C)

This write-only register has an offset of hex 2C.

The byte order of this register is independent of whether the Intel or Motorola register interface is enabled. The following figure shows the Intel and Motorola views of the Direction Steps register.

```
Intel View Of Register

    byte 3        byte 2        byte 1        byte 0

 31          24 23          16 15           8  7           0
┌─────────────┬─────────────┬─────────────┬─────────────┐
│ step code 4 │ step code 3 │ step code 2 │ step code 1 │
└─────────────┴─────────────┴─────────────┴─────────────┘


Motorola View Of Register

    byte 0        byte 1        byte 2        byte 3

 31          24 23          16 15           8  7           0
┌─────────────┬─────────────┬─────────────┬─────────────┐
│ step code 1 │ step code 2 │ step code 3 │ step code 4 │
└─────────────┴─────────────┴─────────────┴─────────────┘
```

Figure 3-125. Direction Steps Register, Offset Hex 2C

This register is used to specify up to four draw and step codes to the coprocessor, and to initiate a draw and step operation.

Writing data to byte 3 of this register initiates a draw and step operation. A draw and step operation can be initiated by a single 32-bit access, by two 16-bit accesses where bytes 2 and 3 are written last, or by four 1-byte accesses where byte 3 is written last. If multiple draw and step operations are required with the same draw and step codes, the operation can be initiated by writing to byte 3.

Before initiating a draw and step operation, the PEL Operation register must be configured to set up the data path and flags for the draw and step operation. See "Draw and Step" on page 3-95 for full details.

Foreground Mix Register (Offset 48)

This write-only register has an offset of hex 48.

```
  7    6    5    4    3    2    1    0
+---------------------------------------+
|            Foreground Mix             |
+---------------------------------------+
```

Figure 3-126. Foreground Mix Register, Offset Hex 48

The Foreground Mix register (bits 7 – 0) holds the foreground mix value that specifies a logic or arithmetic function to be performed between the destination and function 1 second operand PELs, during an operation where the pattern PEL value is 1.

See “Logical and Arithmetic Functions” on page 3-109 for details and mix functions available.

Background Mix Register (Offset 49)

This write-only register has an offset of hex 49.

```
  7    6    5    4    3    2    1    0
+---------------------------------------+
|            Background Mix             |
+---------------------------------------+
```

Figure 3-127. Background Mix Register, Offset Hex 49

The Background Mix register (bits 7 – 0) holds the background mix value that specifies a logic or arithmetic function to be performed between the destination and function 0 second operand PELs during an operation where the pattern PEL value is 0.

See “Logical and Arithmetic Functions” on page 3-109 for details and mix functions available.

Destination Color Compare Condition Register (Offset 4A)

This write-only register has an offset of hex 4A.

7	6	5	4	3	2	1	0
–	–	–	–	–	DCC		

```
  – : Set to 0
DCC : Destination Color Compare Condition
```

Figure 3-128. Destination Color Compare Condition Reg, Offset Hex 4A

The register field is defined as follows:

DCC The Destination Color Compare Condition field (bits 2–0) specifies the destination color compare condition when the destination update is inhibited.

The DCC field definitions are shown in the following figure.

DCC Field (binary)	Destination Color Compare Condition
0 0 0	Always True (disable update)
0 0 1	Destination > Color Compare Value
0 1 0	Destination = Color Compare Value
0 1 1	Destination < Color Compare Value
1 0 0	Always False (enable update)
1 0 1	Destination > = Color Compare Value
1 1 0	Destination < > Color Compare Value
1 1 1	Destination < = Color Compare Value

Figure 3-129. Destination Color Compare Condition Bit Definition

Destination Color Compare Value Register (Offset 4C)

This write-only register has an offset of hex 4C.

```
 31                                                      0
┌─────────────────────────────────────────────────────────┐
│             Destination Color Compare Value             │
└─────────────────────────────────────────────────────────┘
```

Figure 3-130. Destination Color Compare Value Register, Offset Hex 4C

The Destination Color Compare Value register (bits 31 – 0) contains the comparison value for the destination PELs to be compared when color compare is enabled. Only the corresponding number of bits-per-PEL in the destination are required in this register (for example, if the destination is 4 bits-per-PEL, only the 4 low-order bits of this register are used). The bits of this register that are more significant than the number of bits-per-PEL need not be written.

See "Color Compare" on page 3-113 for details of the color compare function.

PEL Bit Mask (Plane Mask) Register (Offset 50)

This write-only register has an offset of hex 50.

```
31                                                 0
+---------------------------------------------------+
|                   PEL Bit Mask                    |
+---------------------------------------------------+
```

Figure 3-131. PEL Bit Mask (Plane Mask) Register, Offset Hex 50

The PEL Bit Mask register (bits 31 – 0) determines the bits within each PEL that are subject to update by the coprocessor. A 1 means the corresponding bit is enabled for updates. A 0 means the corresponding bit is not updated.

A bit that is not write enabled is prevented from affecting either arithmetic operations or the destination color compare comparison, so masked bits are excluded from the operation or comparison. Only the corresponding number of bits-per-PEL in the destination are required in this register (for example, if the destination is 4 bits-per-PEL, only the 4 low-order bits of this register are used). The bits of this register that are more significant than the number of bits-per-PEL need not be written.

See "PEL Bit Masking" on page 3-113 for details of the PEL bit mask function.

Carry Chain Mask Register (Offset 54)

This write-only register has an offset of hex 54.

```
 31                                                      0
┌──────────────────────────────────────────────────────────┐
│                    Carry Chain Mask                      │
└──────────────────────────────────────────────────────────┘
```

Figure 3-132. Carry Chain Mask Field Register, Offset Hex 54

The Carry Chain Mask register (bits 31 – 0) contains a mask up to 31 bits wide. The mask is used to specify how the carry chain of the coprocessor is propagated when performing arithmetic update mixes and color compare operations.

A 0 in the mask means that the carry out of this bit-position of the coprocessor is not to be propagated to the next significant bit-position. A 1 in the mask means that propagation is to take place. The PEL value can be split into sections within the PEL.

Only the corresponding number of bits-per-PEL in the destination are required in this register (for example, if the destination is 4 bits-per-PEL, only the 4 low-order bits of this register are used). The bits of this register that are more significant than the number of bits-per-PEL, need not be written. There is no carry out of the most-significant bit of the PEL irrespective of the setting of the corresponding carry chain mask bit.

See "Breaking the Coprocessor Carry Chain" on page 3-111 for details on the carry chain function.

Foreground Color Register (Offset 58)

This write-only register has an offset of hex 58.

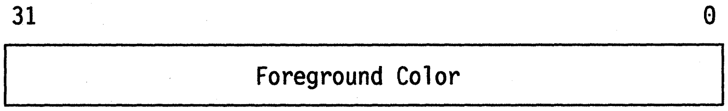


Figure 3-133. Foreground Color Register, Offset Hex 58

The Foreground Color register (bits 31 – 0) holds the foreground color to be used during coprocessor operations. The foreground color can be specified as the foreground source by setting up the appropriate field in the PEL Operations register.

Only the corresponding number of bits-per-PEL in the destination are required in this register (for example, if the destination is 4 bits-per-PEL, only the 4 low-order bits of this register are used). The bits of this register that are more significant than the number of bits-per-PEL need not be written.

Background Color Register (Offset 5C)

This write-only register has an offset of hex 5C.

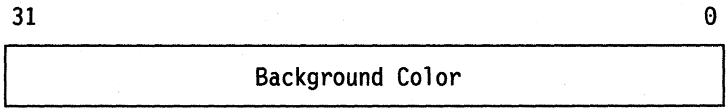


Figure 3-134. background Color Register, Offset Hex 5C

The Background Color register (bits 31 – 0) holds the background color to be used during coprocessor operations. The background color can be specified as the background source by setting up the appropriate field in the PEL Operation register.

Only the corresponding number of bits-per-PEL in the destination are required in this register (for example, if the destination is 4 bits-per-PEL, only the 4 low-order bits of this register are used). The bits of this register that are more significant than the number of bits-per-PEL need not be written.

Operation Dimension 1 Register (Offset 60)

This write-only register has an offset of hex 60.

```
 31                                                       0
┌──────────────────────────────────────────────────────────┐
│                  Operation Dimension 1                   │
└──────────────────────────────────────────────────────────┘
```

Figure 3-135. Operation Dimension 1 Register, Offset Hex 60

The Operation Dimension 1 register (bits 31 – 0) specifies the width of the rectangle to be drawn by the PxBlt function, or the length of line in a line draw operation. The value is an unsigned quantity, and must be one less than the required width. To draw a line 10 PELs long, the value 9 must be written to this register.

The value written to this register must be within the range (0 to 4095).

Operation Dimension 2 Register (Offset 62)

This write-only register has an offset of hex 62.

```
 31                                                       0
┌──────────────────────────────────────────────────────────┐
│                  Operation Dimension 2                   │
└──────────────────────────────────────────────────────────┘
```

Figure 3-136. Operation Dimension 2 Register, Offset Hex 62

The Operation Dimension 2 register (bits 31 – 0) specifies the height of the rectangle to be drawn by the PxBlt function. The value is an unsigned quantity, and must be one less than the required height. To draw a rectangle 10 PELs high, the value 9 must be written to this register.

The value written to this register must be within the range (0 to 4095).

Mask Map Origin X Offset Register (Offset 6C)

This write-only register has an offset of hex 6C.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mask Map Origin X Offset															

Figure 3-137. Mask Map Origin X Offset Register, Offset Hex 6C

The Mask Map Origin X Offset register (bits 15 – 0) specifies the X offset of the mask map origin, relative to the origin of the destination map.

The value written to this register must be within the range (0 to 4095).

Mask Map Origin Y Offset Register (Offset 6E)

This write-only register has an offset of hex 6E.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mask Map Origin Y Offset															

Figure 3-138. Mask Map Origin Y Offset Register, Offset Hex 6E

The Mask Map Origin Y Offset register (bits 15 – 0) specifies the Y offset of the mask map origin, relative to the origin of the destination map.

The value written to this register must be within the range (0 to 4095).

Source X Address Register (Offset 70)

This read/write register has an offset of hex 70.

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
┌───────────────────────────────────────────────┐
│                Source X Address               │
└───────────────────────────────────────────────┘
```

Figure 3-139. Source X Address Register, Offset Hex 70

The Source X Address register (bits 15 – 0) specifies the X coordinate of the coprocessor operation source PEL.

The value written to this register must be within the range (0 to 4095).

Source Y Address Register (Offset 72)

This read/write register has an offset of hex 72.

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
┌───────────────────────────────────────────────┐
│                Source Y Address               │
└───────────────────────────────────────────────┘
```

Figure 3-140. Source Y Address Register, Offset Hex 72

The Source Y Address register (bits 15 – 0) specifies the Y coordinate of the coprocessor operation source PEL.

The value written to this register must be within the range (0 to 4095).

Pattern X Address Register (Offset 74)

This read/write register has an offset of hex 74.

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
+------------------------------------------------+
|                Pattern X Address               |
+------------------------------------------------+
```

Figure 3-141. Pattern C Address Register, Offset Hex 74

The Pattern X Address register (bits 15 – 0) specifies the X coordinate of the coprocessor operation pattern PEL.

The value written to this register must be within the range (0 to 4095).

Pattern Y Address Register (Offset 76)

This read/write register has an offset of hex 76.

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
+------------------------------------------------+
|                Pattern Y Address               |
+------------------------------------------------+
```

Figure 3-142. Pattern Y Address Register, Offset Hex 76

The Pattern Y Address register (bits 15 – 0) specifies the Y coordinate of the coprocessor operation pattern PEL.

The value written to this register must be within the range (0 to 4095).

Destination X Address Register (Offset 78)

This read/write register has an offset of hex 78.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Destination X Address															

Figure 3-143. Destination X Address Register, Offset Hex 78

The Destination X Address register (bits 15 – 0) specifies the X coordinate of the coprocessor operation destination PEL.

This register must be written as a 16-bit sign-extended twos complement number in the range (–2048 to 6143).

Destination Y Address Register (Offset 7A)

This read/write register has an offset of hex 7A.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Destination Y Address															

Figure 3-144. Destination Y Address Register, Offset Hex 7A

The Destination Y Address register (bits 15 – 0) specifies the Y coordinate of the coprocessor operation destination PEL.

This register must be written as a 16-bit sign-extended twos complement number in the range (–2048 to 6143).

PEL Operations Register (Offset 7C)

This write-only register has an offset of hex 7C.

The PEL Operations register is used to define the flow of data during an operation. It specifies the address update function that is to be performed, and initiates PxBlt and line draw operations.

The contents of the PEL Operation register are preserved throughout an operation.

The byte order of this register is dependent on whether the Intel or Motorola register interface is enabled. The following figure shows the Intel and Motorola views of the PEL Operations register.

```
Intel View Of Register

|   Byte 3      |   Byte 2      |   Byte 1      |   Byte 0      |

31 . . . . . .24 23. . . . . .16 15. . . . . . 8 7 . . . . . . 0
+---+---+-------+-------+-------+-------+-+-+-+-+---+---+-+-----+
| BS| FS| Step  |Source | Dest  | Patt  |-|-|-|-|MSK| DM|-| Oct |
+---+---+-------+-------+-------+-------+-+-+-+-+---+---+-+-----+


Motorola View Of Register

|   Byte 0      |   Byte 1      |   Byte 2      |   Byte 3      |

31 . . . . . .24 23. . . . . .16 15. . . . . . 8 7 . . . . . . 0
+---+---+-+-----+-------+-+-+-+-+-------+-------+---+---+-------+
|MSK| DM|-| Oct | Patt  |-|-|-|-|Source | Dest  | BS| FS| Step  |
+---+---+-+-----+-------+-+-+-+-+-------+-------+---+---+-------+
```

-	:	Set to 0	Dest	:	Destination PEL Map
BS	:	Background Source	Patt	:	Pattern PEL Map
FS	:	Foreground Source	MSK	:	Mask PEL Map
Step	:	Step Function	DM	:	Drawing Mode
Source	:	Source PEL Map	Oct	:	Direction Octant

Figure 3-145. PEL Operations Register, Offset Hex 7C

All operations, with the exception of draw and step (see “Draw and Step” on page 3-95 and “Direction Steps Register (Offset 2C)” on page 3-132), are initiated by writing to the *most-significant byte* of this register. Therefore, an operation can be initiated by a single 32-bit write, two 16-bit writes when bytes 2 and 3 are written last, or four 1-byte accesses where byte 3 is written last.

The fields in the PEL Operation register are shown, by bytes, in the following figures.

Byte 3

7	6	5	4	3	2	1	0
BS		FS		Step			

BS : Background Source
FS : Foreground Source
Step : Step Function

Figure 3-146. PEL Operations Register, Byte 3

BS The Background Source field (byte 3; bits 7, 6) determines the background source that is to be combined with the destination when the pattern PEL equals 0 (background mix).

BS Field (binary)	Background Source
0 0	Background Color
0 1	Reserved
1 0	Source PEL Map
1 1	Reserved

Figure 3-147. PEL Operations Register Background Source Value Assignments

FS The Foreground Source field (byte 3; bits 5, 4) determines the foreground source that is to be combined with the destination when the pattern PEL equals 1 (foreground mix).

FS Field (binary)	Foreground Source
0 0	Foreground Color
0 1	Reserved
1 0	Source PEL Map
1 1	Reserved

Figure 3-148. PEL Operations Register Foreground Source Value Assignments

Step The Step Function field (byte 3; bits 3 – 0) determines how the coprocessor address is modified as PEL data is manipulated. This field can be regarded as the coprocessor function code. Writing to this field starts the coprocessor operation, except for draw and step functions. Draw and step operations are started by writing to the Direction Steps register.

Step Field (binary)	Step Function
0 0 0 0	Reserved
0 0 0 1	Reserved
0 0 1 0	Draw and Step Read
0 0 1 1	Line Draw Read
0 1 0 0	Draw and Step Write
0 1 0 1	Line Draw Write
0 1 1 0	Reserved
0 1 1 1	Reserved
1 0 0 0	PxBlt
1 0 0 1	Inverting PxBlt
1 0 1 0	Area Fill PxBlt
1 0 1 1	Reserved
.	.
.	.
1 1 1 1	Reserved

Figure 3-149. PEL Operations Register Step Function Value Assignments

```
|               Byte 2                |

  7    6    5    4    3    2    1    0
+------------------+------------------+
|      Source      |       Dest       |
+------------------+------------------+

  Source : Source PEL Map
    Dest : Destination PEL Map
```

Figure 3-150. PEL Operations Register, Byte 2

Source The Source PEL Map field (byte 2; bits 7 – 4) determines the location of PEL map source data. The combination of this field and the Foreground and Background Source fields determine the data that is to be used as the source data for coprocessor functions.

Source Field (binary)	Source PEL Map
0 0 0 0	Reserved
0 0 0 1	PEL Map A
0 0 1 0	PEL Map B
0 0 1 1	PEL Map C
0 1 0 0	Reserved
.	.
.	.
1 1 1 1	Reserved

Figure 3-151. PEL Operations Register Source PEL Map Value Assignments

Dest The Destination PEL Map field (byte 2; bits 3 – 0) determines the location of the destination data to be modified during an operation.

Dest Field (binary)	Destination PEL Map
0 0 0 0	Reserved
0 0 0 1	PEL Map A
0 0 1 0	PEL Map B
0 0 1 1	PEL Map C
0 1 0 0	Reserved
.	.
.	.
1 1 1 1	Reserved

Figure 3-152. PEL Operations Register Destination PEL Map Value Assignments

```
|              Byte 1               |

  7    6    5    4    3    2    1    0
+-------------------+----+----+----+----+
|       Patt        | -  | -  | -  | -  |
+-------------------+----+----+----+----+

       - : Set to 0
    Patt : Pattern PEL Map
```

Figure 3-153. PEL Operations Register, Byte 1

Patt The Pattern PEL Map field (byte 1; bits 7−4) determines the pattern data to be used during an operation. Code 1000 causes the coprocessor to assume that the pattern is 1 across the whole operation, and to use the foreground function on all PELs. This effectively turns off the use of the pattern. Code 1001 causes the pattern to be generated from source data. Every 0 PEL in the source generates a background pattern PEL; every nonzero PEL in the source generates a foreground pattern PEL.

Patt Field (binary)	**Pattern PEL Map**
0 0 0 0	Reserved
0 0 0 1	PEL map A
0 0 1 0	PEL map B
0 0 1 1	PEL map C
0 1 0 0	Reserved
0 1 0 1	Reserved
0 1 1 0	Reserved
0 1 1 1	Reserved
1 0 0 0	Foreground (fixed)
1 0 0 1	Generated from Source
1 0 1 0	Reserved
.	.
.	.
1 1 1 1	Reserved

Figure 3-154. PEL Operations Register Pattern PEL Map Value Assignments

```
|              Byte 0              |

   7    6    5    4    3    2    1    0
+---------+---------+----+-------------+
|   MSK   |   DM    | -  |     Oct     |
+---------+---------+----+-------------+

     - : Set to 0         DM : Drawing Mode
   MSK : Mask PEL Map    Oct : Direction Octant
```

Figure 3-155. PEL Operations Register, Byte 0

MSK The Mask PEL Map field (byte 0; bits 7, 6) determines how the mask map is used. See "Scissoring with the Mask Map" on page 3-90 for details of the mask map modes.

MSK Field (binary)	Mask PEL Map
0 0	Mask Map Disabled
0 1	Mask Map Boundary Enabled
1 0	Mask Map Enabled
1 1	Reserved

Figure 3-156. PEL Operations Register Mask PEL Map Value Assignments

DM The Drawing Mode field (byte 0; bits 5, 4) determines the attributes of line draw and draw and step operations.

DM Field (binary)	Drawing Mode
0 0	Draw All PELs
0 1	Draw First PEL Null
1 0	Draw Last PEL Null
1 1	Draw Area Boundary

Figure 3-157. PEL Operations Register Drawing Mode Value Assignments

Oct The Direction Octant field (byte 0; bits 2 – 0) is comprised of three bits, DX, DY, and DZ.

```
+--------------+
|     Oct      |
|  2    1    0 |
+----+----+----+
| DX | DY | DZ |
+----+----+----+
```

Figure 3-158. PEL Operations Register Direction Octant Values

XGA System Interface

Multiple Instances

Up to eight Instances of an XGA subsystem can be installed in a system unit. Addressing the I/O registers, memory-mapped registers, and video memory for each Instance is controlled by the contents of the XGA POS registers. See "XGA POS Registers" on page 3-151 for more information.

Multiple XGA Subsystems in VGA Mode

The VGA has only one set of addresses allocated to it. It is not possible to have multiple XGA subsystems in VGA mode responding to update requests simultaneously. However, more than one XGA subsystem can be in VGA mode if only one has VGA address decoding enabled using the Operating Mode register. Subsystems with VGA address decoding disabled continue to display the correct picture. See "Adapter Coexistence with VGA" on page 3-170 for further information.

Note: The XGA *must not* be disabled using the Subsystem Enable field in XGA POS Register 2.

Multiple XGA Subsystems in 132-Column Text Mode

In 132-column text mode, the XGA responds to VGA address decodes, and the same rules apply as for multiple XGA subsystems in VGA mode. See "Adapter Coexistence with VGA" on page 3-170 for further information.

Multiple XGA Subsystems in Extended Graphics Mode

Extended Graphics modes are controlled by a bank of 16 I/O registers that are located in one of eight possible locations. Up to eight XGA subsystems can be installed in a system unit. Each Instance of XGA installed is positioned at a unique I/O and memory location, so each can be used independently in the system. See "Adapter Coexistence with Other XGA Subsystems" on page 3-170 for details on controlling multiple XGAs.

The XGA coprocessor memory-mapped registers occupy a bank of 128 contiguous register addresses that are mapped in memory space. These registers can also be relocated, allowing up to eight Instances of the XGA coprocessor to coexist in a system.

The locations of these registers are controlled by the XGA POS registers. See “XGA POS Registers” on page 3-151 for the register details, and see “Locating the XGA Subsystem” on page 3-171 for programming considerations on reading and using the data contained in them.

XGA POS Registers

The XGA subsystem has movable I/O addresses for the display controller, allowing more than one XGA subsystem to be installed in a system unit.

All POS registers detailed in this section are set up during system configuration and must never be written. All registers are specified relative to a base address. Details of how to locate the base address and read the registers are given in “Locating the XGA Subsystem” on page 3-171.

Register Usage Guidelines

Unless otherwise stated, the following are guidelines to be used when accessing the POS registers:

- All registers are 8 bits long.
- All registers are read only.
- Special reserved register bits must be used as follows:
  - Register bits marked with ‘—’ must be masked off if the contents of the register is to be tested.
  - Register bits marked with ‘#’ are reserved and the state of these bits must be preserved. This register must be masked off if the contents of the register is to be tested.

Subsystem Identification Low Byte Register (Base + 0)

This read-only register is located at base address + 0. *Do not write to* this register. When read, this register returns the low byte of the POS ID. See “Reading POS Data” on page 3-171 for more information.

Subsystem Identification High Byte Register (Base + 1)

This read-only register is located at base address + 1, *do not write to* this register. When read, this register returns the high byte of the POS ID. See “Reading POS Data” on page 3-171 for more information.

POS Register 2 (Base + 2)

This read-only register is located at base address + 2. *Do not write to* this register.

7	6	5	4	3	2	1	0
ROM				INST			EN

```
 ROM : ROM Address
INST : Instance
  EN : Subsystem Enable
```

Figure 3-159. POS Register 2, Base Address + 2

The fields in this register are defined as follows:

ROM The ROM Address field (bits 7–4) specifies which of 16 possible 8KB memory locations has been assigned to the XGA ROM. The ROM occupies the first 7KB of this 8KB block; the other 1KB is occupied by the coprocessor memory-mapped registers.

The Instance field specifies the 128-byte section within this 1KB block that is allocated to the subsystem. See the following figure. For example, Instance 2 has its coprocessor registers located in the third 128-byte section of the 1KB block.

ROM Address		Coprocessor Register Base Address (hex)							
Field	Range (hex)	Instance: 0	1	2	3	4	5	6	7
0000	C0000 : C1BFF	C1C00	C1C80	C1D00	C1D80	C1E00	C1E80	C1F00	C1F80
0001	C2000 : C3BFF	C3C00	C3C80	C3D00	C3D80	C3E00	C3E80	C3F00	C3F80
0010	C4000 : C5BFF	C5C00	C5C80	C5D00	C5D80	C5E00	C5E80	C5F00	C5F80
0011	C6000 : C7BFF	C7C00	C7C80	C7D00	C7D80	C7E00	C7E80	C7F00	C7F80
0100	C8000 : C9BFF	C9C00	C9C80	C9D00	C9D80	C9E00	C9E80	C9F00	C9F80
0101	CA000 : CBBFF	CBC00	CBC80	CBD00	CBD80	CBE00	CBE80	CBF00	CBF80
0110	CC000 : CDBFF	CDC00	CDC80	CDD00	CDD80	CDE00	CDE80	CDF00	CDF80
0111	CE000 : CFBFF	CFC00	CFC80	CFD00	CFD80	CFE00	CFE80	CFF00	CFF80
1000	D0000 : D1BFF	D1C00	D1C80	D1D00	D1D80	D1E00	D1E80	D1F00	D1F80
1001	D2000 : D3BFF	D3C00	D3C80	D3D00	D3D80	D3E00	D3E80	D3F00	D3F80
1010	D4000 : D5BFF	D5C00	D5C80	D5D00	D5D80	D5E00	D5E80	D5F00	D5F80
1011	D6000 : D7BFF	D7C00	D7C80	D7D00	D7D80	D7E00	D7E80	D7F00	D7F80
1100	D8000 : D9BFF	D9C00	D9C80	D9D00	D9D80	D9E00	D9E80	D9F00	D9F80
1101	DA000 : DBBFF	DBC00	DBC80	DBD00	DBD80	DBE00	DBE80	DBF00	DBF80
1110	DC000 : DDBFF	DDC00	DDC80	DDD00	DDD80	DDE00	DDE80	DDF00	DDF80
1111	DE000 : DFBFF	DFC00	DFC80	DFD00	DFD80	DFE00	DFE80	DFF00	DFF80

Figure 3-160. XGA ROM, Memory-Mapped Register Assignments

INST The value in this field indicates the *Instance* of this XGA subsystem. Coexisting XGA Subsystems each have unique Instance values.

The Instance field (bits 3 – 1) specifies the set of I/O addresses allocated to the display controller registers. See Figure 3-161. The lowest address of each set of addresses is referred to as the I/O base address.

INST Field (binary)	Instance Number	Instance I/O Base Address (hex)
000	0	2100
001	1	2110
010	2	2120
011	3	2130
100	4	2140
101	5	2150
110	6	2160
111	7	2170

Figure 3-161. I/O Device Address Bit Assignment

EN The Subsystem Enable field (bit 0) indicates whether the subsystem is enabled. When read as 1, the subsystem is enabled for address decoding for all non-POS addresses. When read as 0, only POS registers can be accessed; all other accesses to the subsystem have no effect.

POS Register 4 (Base + 4)

This read-only register is located at base address + 4. *Do not write to* this register.

7	6	5	4	3	2	1	0
Video Memory Base Address							VE

VE : Video Memory Enable

Figure 3-162. POS Register 4, Base Address + 4

The fields in this register are defined as follows:

Video Memory Base Address
This field (bits 7 – 1) contains the most significant 7 bits of the address where the XGA memory is located. Three more bits are provided by the Instance in POS byte 1. This gives a video memory base address on a 4MB boundary. See the following figure.

31 - 25	24 - 22	21 - 0
Video Memory Base Address	Instance 0 – 7	4MB of Addressable Memory

Figure 3-163. XGA Video Memory Base Address.

For example, if the video memory base address is set to 1 and Instance 6 has been selected, the XGA video memory is located at hex 03800000.

VE When the Video Memory Enable field (bit 0) is set to 0, the 4MB aperture is disabled; when set to 1, the 4MB aperture is enabled.

POS Register 5 (Base + 5)

This read-only register is located at base address + 5. *Do not write to* this register.

7	6	5	4	3	2	1	0
–	–	–	–	AP1			

```
   – : Undefined On Read
 AP1 : 1MB Aperture Base Address
```

Figure 3-164. POS Register 5, Base Address + 5

The field in this register is defined as follows:

AP1 The 1MB Aperture Base Address field (bits 3 – 0) indicates where the 1MB aperture is placed in system address space, or if the aperture has been disabled. The following figure describes the use of this field.

AP1 Field (binary)	1MB Aperture Location (hex)
0000	Disabled
0001	00100000
0010	00200000
0011	00300000
0100	00400000
0101	00500000
0110	00600000
0111	00700000
1000	00800000
1001	00900000
1010	00A00000
1011	00B00000
1100	00C00000
1101	00D00000
1110	00E00000
1111	00F00000

Figure 3-165. 1MB Aperture Base Address Value Assignments

Virtual Memory Description

The XGA coprocessor can address either real or virtual memory. When addressing real memory, the linear address calculated by the coprocessor is passed directly to the system microprocessor or local video memory. When addressing virtual memory, the linear address from the coprocessor is translated by on-chip virtual memory translation logic before the translated address is passed to the system microprocessor or local video memory. Virtual address translation is enabled or disabled by a control bit in the XGA.

The coprocessor uses two levels of tables to translate the linear address from the coprocessor to a physical address. Addresses are translated through a page directory and page table to generate a physical address to memory pages that are 4KB in size. The page directory and page tables are of the same form as those used by the 80386 Processor Paging Unit.

Address Translation

The linear address from the coprocessor is divided into three fields that are used to look up the corresponding physical address. The fields, called directory index, table index, and offset, are illustrated in the following figure.

31 22	21 12	11 0
Directory Index	Table Index	Offset

Figure 3-166. Linear Address Fields

The location of the page directory is at a fixed physical address in memory that must be on a page (4KB) address boundary. The coprocessor has a Page Directory Base Address register that must be loaded with the address of the page directory base.

The translation process is illustrated in Figure 3-167 on page 3-157.

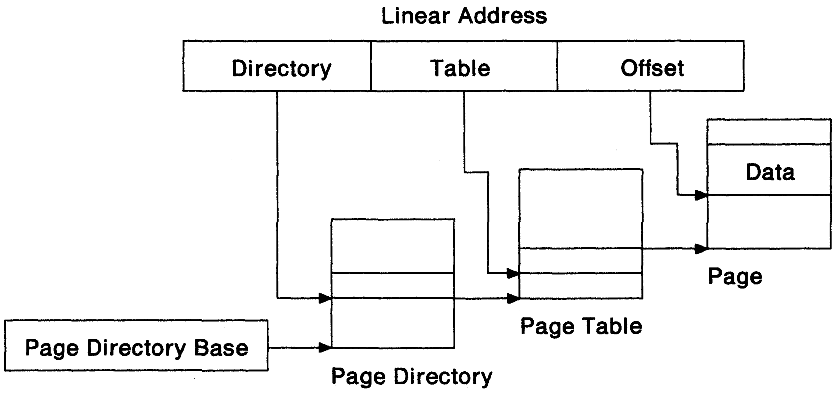


Figure 3-167. Linear to Physical Address Translation

The Directory Index field of the linear address is used to index into the page directory. The entry read from the page directory contains a 20-bit page table address and some statistical information in the low order bits.

The 20-bit page table address points to the base of a page table in memory. The Table Index field in the linear address is used to index into the page table. The entry read from the page table contains a 20-bit page address and some statistical information in the low order bits.

The 20-bit page address points to the base of a 4KB page in memory. The Offset field in the linear address is used to index into the page. The entry read from the page contains the data required by the memory access.

Page Directory and Page Table Entries

The entries of the page directory and page table are very similar. The format of an entry is shown in the following figure.

31 12		6	5		2	1	0
Page Table/Page Addr		D	A		U/S	R/W	P

D	- Dirty Bit
A	- Accessed Bit
U/S	- User/Supervisor Bit
R/W	- Read/Write Bit
P	- Present Bit

Figure 3-168. Page Directory and Page Table Entry

The top 20 bits of the entry are either the page table address or the page address. The low order bits are as follows:

Dirty Bit (bit 6): This bit is set before a write to an address covered by that page table entry occurs. The dirty bit is undefined for page directory entries.

Accessed Bit (bit 5): This bit is set for both types of entry before a read or write access occurs to an address covered by the entry.

User/Supervisor and Read/Write Bits (bits 2,3): These bits prevent unauthorized use of page directory and page table entries. Accesses by the coprocessor can be defined as a supervisor or user access, depending on the status of the application using the coprocessor. The access type is defined by a bit in the virtual memory (VM) Control register. If the access is defined as supervisor, no protection is provided and all accesses to the page directory and page tables are permitted.

For a user access, the U/S and R/W bits are checked to ensure that access to that entry is permitted. The meaning of these bits is shown in the following figure.

U/S	**R/W**	**Access Rights of User**
0	0	Access not permitted
0	1	Access not permitted
1	0	Reads permitted, writes not permitted
1	1	Reads and writes permitted

Figure 3-169. Page Directory and Page Table Access Rights in User Mode

Present Bit (bit 0): The present bit indicates whether a page directory or page table entry can be used in translation. If the bit is set, it indicates that the page table or page that the entry refers to is present in memory.

The XGA Implementation of Virtual Memory

The XGA coprocessor operates with a page directory and page tables in the format described. The coprocessor has its own internal cache of translated addresses to avoid it having to perform the two-stage translation process on every coprocessor access. This cache is referred to as a translate look-aside buffer.

The Translate Look-aside Buffer

The translate look-aside buffer (TLB) has two entries, one entry for the source, pattern, and mask PEL maps, and another for the destination PEL map, as shown in Figure 3-170. Each entry is reserved specifically for use by one of these maps. Each entry in the TLB contains the top 20 bits of a linear address (the address tag), an entry valid flag bit, and the top 20 bits of the physical address (the real page address) corresponding to that linear address. When a linear address is passed from the coprocessor to the virtual address hardware, the top 20 bits of the linear address are first compared against the appropriate TLB entry address tag. If they match and the TLB entry flag bit is valid, the real page address in that TLB is used as the top 20 bits of the physical address for that access. The bottom 12 bits of the physical address are provided from the bottom 12 bits of the linear address (the offset).

		31 12	31 12
Source, Pattern, and Mask Entry	V	Linear Address Tag	Real Page Address
Destination Entry	V	Linear Address Tag	Real Page Address

V - Valid Bits

Figure 3-170. Translate Look-Aside Buffer

If the linear address from the coprocessor matches the address tag in the TLB for the particular map in use, the access is said to have caused a TLB hit. If the tag does not match, a TLB miss occurs.

The TLB contents are cleared by the hardware under the following circumstances:

- Whenever the Page Directory Base Address register is written
- Whenever a coprocessor operation is suspended (by setting the suspend operation bit in the Coprocessor Control register). See "Coprocessor Control Register (Offset 11)" on page 3-122.

TLB Misses

If a TLB miss occurs, the coprocessor automatically performs the two-level translation required to form the required page address. The contents of the XGA Page Directory Base Address register are used to access the appropriate page directory entry that is used to access the appropriate page table entry. The real page address, resulting from the translation process, is stored in the TLB for use by subsequent accesses that address the same page.

Memory access performed by the coprocessor can be categorized as follows:

Read accesses	Performed on the source, pattern, mask, and destination maps.
Write accesses	Performed only on the destination map.

When the virtual memory hardware accesses the page directory and page tables for a TLB miss, it examines and updates the flags in the low order bits of the entries, as follows:

Accessed Bit (bit 5): Any access (read or write) sets this bit in both the page directory and page table entries.

Dirty Bit (bit 6): Write accesses set the dirty bit in the page table entry. The dirty bit is undefined in page directory entries.

User/Supervisor and Read/Write Bits (bits 2, 3): These bits are examined by the coprocessor. The coprocessor has a bit, programmed by the host operating system, to indicate whether it is being used by a supervisor or user. In user mode the coprocessor determines whether access is permitted, depending on the state of the user/supervisor and the read/write bits in the page directory and page table entries. If access is not permitted, the coprocessor sends a VM-protection-violation interrupt to the system microprocessor and terminates the access cycle. The host operating system must take the appropriate action to recover from the protection-violation interrupt.

Present Bit (bit 0): The coprocessor examines the present bit of the page directory and page table entries. If this bit is not set, it indicates that the page table or page corresponding to that entry may not be resident in memory, and the XGA sends a VM page-not-present interrupt to the system microprocessor. The host operating system fetches the page table or page and places it in memory. The access can then be completed.

Remaining Page Directory or Page Table Entry Bits: The coprocessor ignores all the other bits in the page directory or page table entry. The coprocessor does not modify these bits; they can be used by the operating system. It is advisable to keep entries in the same format as the 80386 page directory and page table entry formats; therefore the Intel rules on the use of the remaining bits must be followed.

System Coherency

In any virtual memory system where more than one device is accessing virtual memory contents, problems can arise over coherency. It is essential that one device does not corrupt the tables or pages of the other device, and the tables and TLBs are kept coherent (in step) with the physical allocation of storage. The hardware mechanism provided by the coprocessor is sufficient to implement coherent virtual memory systems, but care should be taken to avoid coherency problems.

In particular, it is recommended that the 80386 and the coprocessor do not share page directories or page tables. Pages must not be marked as present unless they are locked in place in memory. This maintains coherency between TLB entries in the coprocessor and the true current allocation of real memory. It prevents the operating system from moving these pages out of memory while the coprocessor is accessing them.

VM Page-Not-Present Interrupts

When the coprocessor detects that a page table or page is not present, it sends a page-not-present interrupt to the system microprocessor. The system microprocessor operating system then fetches the required page table or page (usually from disk) and places it in memory. The system microprocessor can determine the faulting address by reading the Current Virtual Address register. After the required page table or page has been fetched, the operating system restarts the faulting memory access by clearing the page-not-present interrupt bit in the XGA. This causes the hardware to retry the access to the faulted entry.

The system microprocessor operating system will probably switch tasks when receiving a page-not-present interrupt. In this case it suspends the coprocessor operation in the normal way (see "Suspending Coprocessor Operations" on page 3-116) before clearing the page-not-present interrupt. The coprocessor state is then saved. When the task where the page-not-present interrupt occurred is restarted, the coprocessor state is restored, the required page table or page is placed in memory, the interrupt is cleared, and the coprocessor operation is restarted. The interrupt must be cleared before the coprocessor operation is restarted, otherwise further interrupts can be lost.

VM Protection-Violation Interrupts

If the coprocessor is directed to access tables or pages that are not permitted (as defined by the user/supervisor and read/write entry bits), the coprocessor generates a protection-violation interrupt. This indicates that something is wrong with the virtual memory system (as set up by operating system software), or that the coprocessor has been programmed incorrectly.

The operating system will probably terminate the coprocessor operation and possibly terminate the faulting task. The coprocessor operation can be terminated by writing to a control bit in the Coprocessor Control register. The coprocessor responds in a similar manner for protection-violation interrupts as it does for page not present interrupts; clearing the interrupt causes the hardware to retry the memory access. To avoid a repeated interrupt, the coprocessor operation must be terminated before the protection-violation interrupt is cleared.

The XGA in Segmented Systems

In a segmented system design, all memory is allocated in blocks called segments. Memory within a segment is guaranteed to be contiguous, and can therefore be addressed directly by the coprocessor using physical addresses (VM is turned off). The segment must be locked in place before any coprocessor operation to ensure that the operating system does not reuse the memory during the operation.

When using 16-bit addressing in the 80386 (for example, under the OS/2 version 1.3 operating system), it is not possible to define a segment of more than 64KB. If the coprocessor data in system memory is restricted to no more than 64KB in length, a single segment can be used and the coprocessor can directly address the data using physical addresses.

When using the OS/2 version 1.3 operating system, larger areas of memory can be requested, but are given in blocks of 64KB (maximum) that are unlikely to be contiguous in real memory. If larger areas in system memory are required, the driving software can turn on the coprocessor VM address translation and perform its own memory management using memory allocated to it by the operating system.

Virtual Memory Registers

The following registers provide virtual memory support for the PEL interface.

Page Directory Base Address Register (Coprocessor Registers, Offset 0)

This write-only register has coprocessor registers offset of hex 0.

```
 31                              12 11                    0
┌──────────────────────────────────┬────────────────────────┐
│ Page Directory Base Addr Pointer │           -            │
└──────────────────────────────────┴────────────────────────┘

        - : Set all bits in field to 0
```

Figure 3-171. Page Directory Base Address Register, Offset Hex 0

The field in this register is defined as follows:

Page Directory Base Addr Pointer
This field (bits 31 – 12) is a 20-bit pointer to the page in physical memory containing the current page directory for the current task.

Loading this register clears the translate look-aside buffer.

Note: This register can be loaded only after the XGA is put in supervisor mode.

Current Virtual Address Register (Coprocessor Registers, Offset 4)

This read-only register has coprocessor registers offset of hex 4. *Do not write to* this register.

31 12	11 0
Faulting Page Address	–

– : Set all bits in field to 0

Figure 3-172. Current Virtual Address Register, Offset Hex 4

The field in this register is defined as follows:

Faulting Page Address
After a VM hardware-not-present interrupt or protection interrupt, read this field (bits 31 – 12) to find the fault address page.

Virtual Memory Control Register (I/O Address 21x6)

This read/write register has an I/O address of hex 21x6. The Virtual Memory Control register is directly mapped to I/O address space.

7	6	5	4	3	2	1	0
NPE	PVE	–	–	–	US	–	EV

```
  – : Set to 0, Undefined on Read
NPE : VM Page Not Present Interrupt Enable
PVE : VM Protection Violation Interrupt Enable
 US : User / Supervisor
 EV : Enable Virtual Address Lookup
```

Figure 3-173. Virtual Memory Control Register

The fields in this register are defined as follows:

NPE The VM Page Not Present Interrupt Enable field (bit 7) controls the sending of an interrupt when a VM page not present condition is detected. When set to 1, an interrupt is sent to the system microprocessor when the not present condition is detected. When set to 0, the not present condition does not send an interrupt. In both cases, the contents of the appropriate VM Interrupt Status register status bit are updated when the not present condition is detected.

PVE The VM Protection Violation Interrupt Enable field (bit 6) controls the sending of an interrupt when a VM protection violation condition is detected. When set to 1, an interrupt is sent to the system microprocessor when the protection violation condition is detected. When set to 0, the protection violation condition does not send an interrupt. In both cases the contents of the appropriate VM Interrupt Status register status bit are updated when the protection violation condition is detected.

US The User/Supervisor field (bit 2) indicates the privilege level of the currently-executing task. When set to 0, the executing task is at privilege levels 0, 1, or 2 (a supervisor task). When set to 1, the executing task is at privilege level 3 (a user task).

If set to supervisor (0), no protection checking is performed by the coprocessor on page directory and page table protection bits. If set to user (1), checking is performed, and a protection interrupt is sent if permitted access rights are violated.

EV The Enable Virtual Address Lookup field (bit 0) controls the virtual address translation. Subsequent addresses generated by the PEL interface hardware are looked up in page tables. If this bit is not set:

- Bit maps must be resident and contiguous.
- The PEL map base addresses are physical addresses.
- All addresses generated by the coprocessor are physical addresses.
- Nonpaged operating systems are supported.

Virtual Memory Interrupt Status Register (I/O Address 21x7)

This read/write register has an I/O address of hex 21x7. The Virtual Memory Interrupt Register is directly mapped to I/O address space.

7	6	5	4	3	2	1	0
NPS	PVS	–	–	–	–	–	–

```
  – : Set to 0, Undefined on Read
NPS : VM Page Not Present Interrupt Status
PVS : VM Protection Violation Interrupt Status
```

Figure 3-174. Virtual Memory Interrupt Status Register

The fields in this register are defined as follows:

NPS When a VM page not present condition occurs, the VM Page Not Present Interrupt Status field (bit 7) is automatically set to 1. This bit is reset to 0 by writing a 1 to it. This allows the value just read to be written back to clear the bits that were set. Writing a 0 to this bit has no effect.

Resetting this bit (writing a 1) causes the VM hardware to retry page translation. If this bit is to be reset before the not present condition has been repaired, the coprocessor operation must be suspended or terminated, otherwise another not-present interrupt is generated by the same not present condition.

PVS When a VM protection violation condition occurs, the VM Protection Violation Interrupt Status field (bit 6) is automatically set to 1. This bit is reset to 0 by writing a 1 to it. This lets the value just read to be written back to clear the bits that were set. Writing a 0 to this bit has no effect.

Resetting this bit (writing a 1) causes the VM hardware to retry page translation. If this bit is to be reset before the protection violation condition has been repaired, the coprocessor operation must be suspended or terminated, otherwise another protection-violation interrupt is generated by the same protection violation condition. Most operating systems do not attempt to recover from a protection violation condition. The coprocessor operation causing this condition is terminated.

XGA Programming Considerations

Adapter Coexistence with VGA

Because the VGA uses fixed I/O and memory mapped address spaces, only one VGA can be active at a time in a system. When the XGA subsystem is installed alongside a VGA or another XGA subsystem, only one of the VGA-capable subsystems can be enabled at one time.

An application can use multiple coexisting VGA or XGA subsystems in VGA mode only by alternately disabling and enabling the various VGAs.

Do not enable more than one VGA concurrently. A disabled (or inactive) VGA retains its visible displayed data, and the overall effect is that of a multiple VGA application.

To successively enable and disable multiple coexisting XGA subsystems in VGA mode, use the Operating Mode register (21x0).

Adapter Coexistence with Other XGA Subsystems

Up to eight XGA subsystems can be installed in a system.

Multiple XGA subsystems can coexist in Extended Graphics mode. Each occupies its own separate ranges of I/O and memory space. An application written to exploit multiple XGA subsystems in this mode can access each Instance of the subsystem without enabling and disabling the subsystem between accesses.

To comply with the restriction on VGA coexistence, a multiple display subsystem application must record, on initialization, the XGA subsystem (if any) originally in VGA mode. On termination, only *that* subsystem should be returned to VGA mode.

Locating the XGA Subsystem

Before using the subsystem, locate the subsystem in I/O and memory space by interrogating and interpreting the POS data for the subsystem.

Reading POS Data

Put every adapter in the system, including the system board video subsystem, into setup mode in turn, and examine their POS IDs to locate any XGA adapters in the system.

For option cards, the procedure is described in System Services BIOS call INT 15h, AH = C4h Programmable Option Select in the *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference.*

For the system board video subsystem, a different procedure is necessary. To place the system board video subsystem in setup mode, write hex 0DF to port 94H; to enable it, write hex 0FF to port 94H.

Interrupts must be disabled for the entire period of time that each adapter is in setup mode.

The POS IDs for all adapters in the system must be read and examined to locate all the XGA adapters in the system. The list of POS IDs allocated to the XGA adapter is as follows:

- 8FD8h
- 8FD9h
- 8FDAh
- 8FDBh.

After successfully matching POS IDs, read the remainder of the POS data bytes for that subsystem. This data is used to calculate the location of the XGA subsystem registers and display buffers in I/O and physical system memory address space. Descriptions of the POS data bit assignments are in "XGA POS Registers" on page 3-151. For future compatibility, mask out all reserved and unused POS data bits before using the data for these calculations. (See "Upward Compatibility" on page 3-196).

Address Calculations

See "XGA POS Registers" on page 3-151 for the technical background to the following register and address space calculations.

ROM Address: Calculate the ROM address from POS data as follows:

```
ROM Address = (ROM Address field × hex 2000) + hex 0C0000
```

The ROM Address field is read from POS Register 2, bits 4 to 7.

Coprocessor Registers: The Coprocessor registers are referenced from a base address. This address depends on the Instance (0−7) of the XGA subsystem and the ROM address calculated as shown in "ROM Address." The Coprocessor register base address is calculated as follows:

```
(((128 × Instance) + hex 1C00) + ROM address)
```

The Instance is read from POS Register 2, bits 1 to 3.

For example:

```
Assuming Instance = 6 and ROM address = hex 0C0000, the Coprocessor
Base Address is hex 0C1F00.
```

I/O Registers: The XGA I/O registers are referenced from a base I/O address. The I/O address is calculated as follows:

```
Hex 21x0 (where x is the Instance)
```

The Instance is read from POS Register 2, bits 1 to 3.

The Video Memory Base Address: The Video Memory Base Address is calculated from the Video Memory Base Address field in POS Register 4, and the Instance.

Figure 3-175 on page 3-173 and Figure 3-176 on page 3-173 show how these two values combine to give the video memory base address.

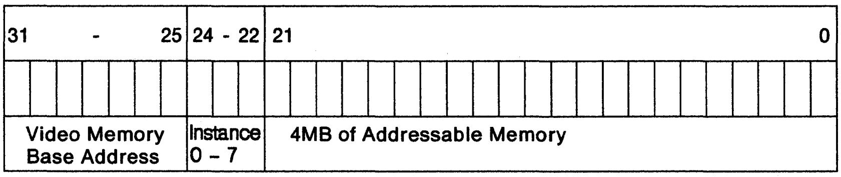


Figure 3-175. XGA Video Memory Base Address

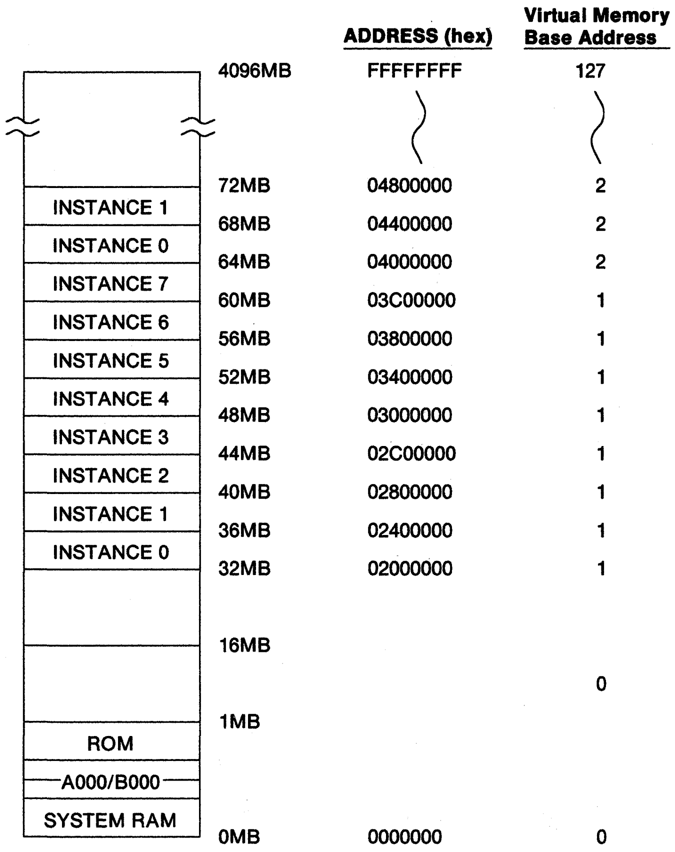


Figure 3-176. The XGA Video Memory Base Address Diagram

The Video Memory Base Address field defines a 32MB address range and the Instance defines a 4MB address range within the 32MB range.

For example:

```
Assuming Instance = 6 and the Video Memory Base Address field = 1,
the Video Memory Base Address is hex 03800000.
```

The video memory base address, when calculated, serves two separate purposes:

4MB System Video Memory Aperture: If enabled (read from bit 0 in POS Register 4 to determine if the aperture is enabled), the 4MB system video memory aperture is located at this address in physical system address space. If virtual addressability to this range of physical address space can be achieved, the entire video memory can be accessed through this aperture at this address.

Video Memory Location in Coprocessor Address Space: The video memory base address has a special significance to the XGA coprocessor. It defines the location of the video memory, including the display PEL map, in the XGA coprocessor's view of system address space. Therefore, the XGA coprocessor recognizes addresses in this range to be addresses in local video memory, rather than general system memory. This is how the XGA coprocessor differentiates video memory from system memory. If an address passed to the XGA coprocessor is in this range, the XGA coprocessor knows that it is operating on a bit map in video memory. If the address is outside this range, the XGA coprocessor assumes it is operating on a bit map in normal system memory, and attempts to use busmastership to access it.

The XGA subsystem operates internally on a 32-bit bus. Therefore, this address is a 32-bit address regardless of whether the XGA subsystem is installed in a 16- or 32-bit slot or system, or whether the 4MB system video memory aperture is enabled or disabled. This applies even on systems where such addresses are not otherwise possible.

1MB Aperture Base Address: The 1MB aperture base address is calculated from the 1MB Aperture Base Address field in POS Register 5, bits 0 to 3.

```
If (1MB Base field ≠ 0)
     1MB Aperture Base Address = 1MB Base field × hex 100000

If (1MB Base field = 0) 1MB Aperture is disabled.
```

Display Type and Video Memory Size

The attached display type is determined by reading the Display ID register (index hex 52). The list of display IDs, including the returned value indicating no attached display, is listed in "Extended Graphics Modes Available" on page 3-180. Depending on the application, it may be necessary to determine the video memory size to ensure that the required mode is available. The method for doing this is also described in "Extended Graphics Modes Available" on page 3-180.

For multiple coexistent XGA subsystems (for a multiple subsystem application), continue until sufficient Instances of the XGA have been located.

VGA Primary Adapter Considerations

Where a single XGA subsystem is providing both VGA and Extended Graphics function, particularly on a system with single display subsystem or monitor, an application using the subsystem in Extended Graphics mode takes on a number of additional systems responsibilities.

Before switching the subsystem into Extended Graphics mode, examine the Operating Mode register, bits 0 and 2, to determine whether the XGA subsystem is enabled in VGA mode or 132-column text mode.

If the XGA subsystem is not enabled in VGA mode, the subsystem is operating as an auxiliary video subsystem and systems messages can be left to the primary VGA source. In this case, the XGA subsystem must not be put into VGA mode unless the current VGA is disabled.

If the XGA subsystem is enabled in VGA mode, the subsystem is the system primary video subsystem, and a number of special considerations apply.

Chaining the INT 10h Video BIOS Handler

The application must chain the INT 10h Video interrupt handler and monitor calls to the INT 10h handler while the application is using the XGA subsystem in Extended Graphics mode.

There are a number of hot-key and error handlers that may attempt to communicate with the VGA while the XGA subsystem is in Extended Graphics mode, so code must be written to handle such calls.

The majority of calls to the INT 10h handler can be ignored (simply return to the caller) while the XGA subsystem is in Extended Graphics mode, but some calls require correct handling.

(Ah) = 00h Set Mode

Set mode calls can come from a critical error or nonmaskable interrupt (NMI) handler. Because failure to restore VGA mode can result in the loss of critical error data or dialogue, applications must allow the mode set operation.

For normal VGA mode setting procedure to occur, the INT 10h handler must restore the subsystem as necessary to VGA mode before chaining on to the next INT 10h interrupt handler.

Note: The NMI handler traditionally issues a return current video state to determine the current mode, followed by a video set mode to the current mode.

If the INT 10h Video interrupt handler of the application detects a video set mode with AL = 7Fh, mode 03h should be substituted after restoring the subsystem to VGA mode.

(Ah) = 0Fh Return current video state

In Extended Graphics mode, the application's INT 10h interrupt handler should return a current mode of 7Fh in AL, to indicate that the subsystem is in a non-VGA mode.

This is a special mode number assigned for this purpose.

INT 24h, Critical Error Handler

The application should trap and revector the DOS critical error handler interrupt vector (INT 24h), as described in the *DOS Technical Reference Manual*. The application is then notified on DOS Critical errors.

The critical error handler of the application should save the video state of the subsystem (as far as necessary), and put the XGA subsystem into VGA mode before chaining on, using the saved vector to the original critical error handler. This lets the dialogue between the critical error handler and the user proceed normally.

After returning from the chained critical error handler, the critical error handler of the application must examine the return code in AL to determine the appropriate action.

0, 1, 3	Control is returned to the application. Put the XGA subsystem back into Extended Graphics mode and restore the video state as necessary.
2	The program is aborted by the system. Leave the XGA subsystem in VGA mode and return.

Alternatively, the application can take over the entire critical error handling dialogue in Extended Graphics mode.

Note: The C language *signal* function can (in some implementations) be used to intercept the critical error handler for this purpose.

INT 23h, Ctrl-Break Exit Address

The application should trap and revector the DOS Ctrl-Break exit address interrupt vector (INT 23h), as described in the *DOS Technical Reference Manual*. The application is notified when the Ctrl-Break key combination is entered.

If the application is not otherwise intercepting Ctrl-Breaks, the XGA subsystem must be put back into VGA mode before chaining on, using the saved vector to the original Ctrl-Break handler. This lets the normal Ctrl-Break handler proceed.

Alternatively, the application can take over the entire Ctrl-Break handling in Extended Graphics mode.

Note: The C language *signal* function can be used to intercept the Ctrl-Break handler for this purpose.

INT 21h, Function 4Ch, Program Terminate Function

The subsystem must be in VGA mode on program termination, regardless of the how the program terminates or is terminated.

To ensure that this is done, the application must trap and revector the normal DOS program terminate function, DOS INT 21h function 4Ch, as described in the *DOS Technical Reference Manual.* On receiving notice of program termination, the application must put the subsystem back into VGA mode and unhook all other hooked interrupt vectors before chaining on for the remainder of program termination handling.

DOS INT 21h function 4Ch is the conventional method used by all programs to terminate. By trapping the DOS function interrupt (INT 21h) and monitoring calls to the program terminate function (4Ch), all routes for a program to terminate normally must be covered.

Note: There are other program terminate functions, including:

- INT 20h
- INT 27h
- INT 21h Function 00h
- INT 21h Function 31h.

For complete coverage, these calls can be revectored and trapped, but they are not used as commonly as the INT 21h Function 4Ch.

All other function calls must be passed to the previous DOS function handler using the saved interrupt vector.

On detecting a call to function 4Ch, put the XGA subsystem into VGA mode before chaining on using the saved vector to the original DOS function handler. This lets the DOS program terminate function proceed normally.

Note: The C language *atexit* function can be used for this purpose.

General Systems Considerations

Coexisting with LIM Expanded Memory Managers

The XGA subsystem uses memory-mapped registers located in the hex C0000/D0000 region of physical address space, as described in "XGA POS Registers" on page 3-151.

This area is used extensively by expanded memory managers to provide expanded memory services to applications.

When the application determines the location of the memory-mapped register space, it must interrogate any expanded memory manager to ensure that there is no contention for this range of physical address space. Use function 25 (Ah) = 58h, get physical address array, as described in the *Lotus/Intel/Microsoft Expanded Memory Specification Version 4.0*. If there is contention, a warning should be issued advising the user to resolve it by use of the expanded memory manager call parameters (usually on the DEVICE= statement in CONFIG.SYS).

INT 2Fh, Screen Switch Notification

For the application to work successfully in multiple virtual DOS machine (MVDM) environments, or in the DOS compatibility box of the OS/2 operating system, it must trap and revector the DOS multiplex vector (2Fh), looking for (Ah) = 40h. Any other values must be passed immediately to the chained INT 2Fh handler. This multiplex interrupt is used with (Ah) = 40h to notify DOS applications of screen switches.

(Al) = 01h DOS mode application being switched to the background.

The application must save its video state and put the display back into VGA mode (if applicable).

(Al) = 02h DOS mode application being switched to the foreground.

The application can switch the subsystem back into Extended Graphics mode, and restore the saved video state.

The range of operations permitted within INT 2Fh processing is limited. For example, it is not permissible to issue disk I/O operations, precluding an entire save and restore of video memory and state. The only way of using this call is for the INT 2Fh interrupt handler to notify the application that a redraw is required (if the application program structure permits).

Extended Graphics Modes Available

The following figure shows the list of modes available according to the display type and size of video memory configured on the XGA subsystem.

Display ID (binary)	Example Displays	Size (in.)	Color	Max Address	512KB Memory	1MB Memory
1111	None				None	None
1101	8503	12	Mono	640x480	640x480x64 Grays	640x480x64 Grays
1110	8513 8512	12 14	Color	640x480	640x480x256 Colors	640x480x256 Colors 640x480x65536 Colors
1011	8515	14	Color	1024x768	640x480x256 Colors 1024x768x16 Colors	640x480x256 Colors 640x480x65536 Colors 1024x768x16 Colors 1024x768x256 Colors
1001	8604 8507	15 19	Mono	1024x768	640x480x64 Grays 1024x768x16 Grays	640x480x64 Grays 1024x768x16 Grays 1024x768x64 Grays
1010	8514	16	Color	1024x768	640x480x256 Colors 1024x768x16 Colors	640x480x256 Colors 640x480x65536 Colors 1024x768x16 Colors 1024x768x256 Colors
0010	8514 compatible	–	Color	1024x768	640x480x256 Colors 1024x768x16 Colors	640x480x256 Colors 640x480x65536 Colors 1024x768x16 Colors 1024x768x256 Colors

Figure 3-177. Availability of Extended Graphics Modes

To determine the display type, read the XGA subsystem register index hex 52, Display ID, and examine the display ID bits returned.

There are two ways to determine the size of video memory installed. Both rely on a write-readback-check, in which a particular value is written to a key location. This value is then read to determine whether the written value has persisted.

- Use the system processor to write a value through an aperture to the word at offset 768KB into video memory. This technique assumes that the system video memory real mode aperture is available. See the sample code in the following figure.

```
;* Assume GS points to start of A0000 Real mode aperture
;* and VGA adapter is in text mode so A0000 Real mode
;* aperture is available for this operation.
;* Where registers are shown as (for instance 21x0h), this should
;* be filled in with the appropriate IO port address after determining
;* the location of the XGA subsystem in IO space
;*
;* First put the adapter PARTIALLY in extended graphics mode
;* to allow use of the system video memory Aperture
        mov   al,0
        mov   dx,21x4h      ; disable XGA interrupts
        out   dx,al
;
        mov   ax,0064h
        mov   dx,21xAh      ; Blank palette
        out   dx,ax         ; indexed XGA register 64h
;
        mov   ax,04h
        mov   dx,21x0h      ; Set adapter in Extended Graphics Mode
        out   dx,al
;
        mov   al,01h
        mov   dx,21x1h      ; Locate video memory Aperture at A0000
        out   dx,al
;
        mov   dx,21x8h      ; System video memory indx reg.
        mov   al,0ch        ; Offset 768K
        out   dx,al         ;
;
        mov   byte ptr gs:[0],0A5h ; Set byte to A5h
        mov   byte ptr gs:[1],0h    ; Avoid shadows on data lines
;
        cmp   byte ptr gs:[0],0A5h  ; Test against value written
        jne   vram_512k      ; 512K video memory only
;
        mov   byte ptr gs:[0],5Ah ; Set byte to 5Ah
        mov   byte ptr gs:[1],0h    ; Avoid shadows on data lines
;
        cmp   byte ptr gs:[0],0A5h  ; Test against value written
        je    vram_1Meg       ; 1 Meg if still matches
        jmp   vram_512k       ; Otherwise 1/2 meg found
```

Figure 3-178. Video Memory Size Determination

- Use the XGA subsystem PxBlt capability to perform a similar test to the previous example; transfer a constant color to the location in video memory, then transfer that value back from video memory to system memory using busmastership.

  This technique works regardless of the availability of a system video memory aperture. However, it requires physical addressability to a location in system memory for the busmastership operation.

Setting the XGA Subsystem Mode

The following points should be observed by *all* software when switching between modes (see "Hardware Considerations" on page 2-11):

- All data in video memory is preserved during a mode switch, provided that the CRT controller is halted at the time using the Display Control 1 register (if switching out of Extended Graphics mode), or the Reset register (if switching out of VGA mode). The CRT controller is described in "CRT Controller" on page 3-19 and "Display Control 1 Register (Index 50)" on page 3-66.
- When switching between VGA modes, the mapping of the VGA memory maps to the video memory is controlled by 2 fields in the following VGA CRT Controller registers:
  - Word/byte mode (CRT Mode Control register, WB field)
  - Doubleword mode (Underline Location register, DW field).

  VGA modes can be split into three groups: byte modes, word modes, and doubleword modes.

  All switches between modes in the same group are indistinguishable from the same mode switches on the VGA.

  Switches between modes in different groups produce different effects from those observed on the VGA. Because the bits controlling the mapping are used for display purposes, the picture is scrambled in both cases.

  Partial mode switches (for example, to load fonts in a text mode) are also possible. The bits used to control the mapping of the data in video memory are used to control the picture display. Therefore, all partial mode switches to update the video memory that do not destroy the picture (and many that do) work correctly.

Individual Mode Setting Procedures

This section gives the register settings to set the subsystem into the various modes available, subject to the rules described in this section. It is important to set the registers in the order they are presented here.

Extended Graphics Mode: To set the XGA subsystem into Extended Graphics mode, the configuration must be capable of supporting the required mode as listed in "Extended Graphics Modes Available" on page 3-180.

XGA Register Name	XGA Reg. ID	Oper	Color Mode Value (hex) 1024x 768x 256	1024x 768x 16	640x 480x 256	640x 480x 65536	Comments
Interrupt Enable	21x4	=	00	00	00	00	Initial Value
Interrupt Status	21x5	=	FF	FF	FF	FF	
Operating Mode	21x0	=	04	04	04	04	Set Extended Graphics Mode
Palette Mask	64	=	00	00	00	00	Blank Display
Video Mem Aperture Ctl	21x1	=	00	00	00	00	Initial Value
Video Mem Aperture Index	21x8	=	00	00	00	00	Initial Value
Virt Mem Ctl	21x6	=	00	00	00	00	Initial Value
Memory Access Mode	21x9	=	03	02	03	04	Initial Value
Disp Mode 1	50	=	01	01	01	01	Prepare for reset
Disp Mode 1	50	=	00	00	00	00	Reset CRT Ctrl
Horiz Total Low	10	=	9D	9D	63	63	)
Horiz Total High	11	=	00	00	00	00	)
Horiz Display End Low	12	=	7F	7F	4F	4F	)
Horiz Display End High	13	=	00	00	00	00	)
Horiz Blank Start Low	14	=	7F	7F	4F	4F	)
Horiz Blank Start High	15	=	00	00	00	00	)
Horiz Blank End Low	16	=	9D	9D	63	63	)
Horiz Blank End High	17	=	00	00	00	00	)
Horiz Sync Start Low	18	=	87	87	55	55	)
Horiz Sync Start High	19	=	00	00	00	00	)
Horiz Sync End Low	1A	=	9C	9C	61	61	)
Horiz Sync End High	1B	=	00	00	00	00	)
Horiz Sync Posn	1C	=	40	40	00	00	)
Horiz Sync Posn	1E	=	04	04	00	00	)
Vert Total Low	20	=	30	30	0C	0C	)
Vert Total High	21	=	03	03	02	02	) XGA CRT
Vert Disp End Low	22	=	FF	FF	DF	DF	) Controller
Vert Disp End High	23	=	02	02	01	01	) param
Vert Blank Start Low	24	=	FF	FF	DF	DF	)
Vert Blank Start High	25	=	02	02	01	01	)
Vert Blank End Low	26	=	30	30	0C	0C	)
Vert Blank End High	27	=	03	03	02	02	)
Vert Sync Start Low	28	=	00	00	EA	EA	)
Vert Sync Start High	29	=	03	03	01	01	)
Vert Sync End	2A	=	08	08	EC	EC	)
Vert Line Comp Low	2C	=	FF	FF	FF	FF	)
Vert Line Comp High	2D	=	FF	FF	FF	FF	)
Sprite Control	36	=	00	00	00	00	Initial Value
Start Addr Low	40	=	00	00	00	00	Initial Value
Start Addr Me	41	=	00	00	00	00	Initial Value
Start Addr High	42	=	00	00	00	00	Initial Value
Buffer Pitch Low	43	=	80	40	50	A0	
Buffer Pitch High	44	=	00	00	00	00	
Clock Sel	54	=	0d	0d	00	00	
Display Mode 2	51	=	03	02	03	04	
Ext Clock Sel	70	=	00	00	00	00	
Display Mode 1	50	=	0F	0F	C7	C7	
Note: Initial Palette loading must be done at this point, by writing to the appropriate XGA subsystem palette/sprite registers. The video memory must also be initialized at this point, to avoid random data appearing when the palette mask is set to make the current display PEL map contents visible.							
Border Color	55	=	00	00	00	00	Initial Value
Palette Mask	64	=	FF	FF	FF	FF	Make visible

Figure 3-179. Extended Graphics Mode Register Settings

VGA Mode: To put the XGA subsystem into VGA mode (subject to the rules discussed in "VGA Primary Adapter Considerations" on page 3-175), perform the following operations:

1. Clear the first 256KB of video memory contents. This avoids screen flash caused by random data being present on switching into VGA mode.
2. Write data to the registers in the following sequence:

Value (hex)	**Oper**	**XGA Reg**	**VGA Reg**	**Comments**
00	=	21x1		Aperture Control register
00	=	21x4		Interrupt disable
FF	=	21x5		Clear interrupts
FF	=	64		Palette Mask register
15	=	50		Enable VFB, prepare for reset
14	=	50		Enable VFB, reset CRT controller
00	=	51		Normal scale factors
04	=	54		Select VGA oscillator
00	=	70		External Clock (VGA)
20	=	2A		Ensure no VSync interrupts
01	=	21x0		Switch to VGA mode
01	=		3C3	Enable VGA address decode

Figure 3-180. VGA Mode Write Sequence

3. Set the number of lines in VGA mode (if required) using Video BIOS INT 10h Ah=12h.
4. Set the required VGA mode using Video BIOS INT 10h Ah=00h, set mode.

The XGA subsystem is now in VGA mode.

132-Column Text Mode: The 132-column text mode may be supported in some XGA subsystems and system units. Before directly setting the 132-column text mode, issue a Video BIOS INT 10h Return Functionality State Information call. Examine the list of BIOS supported modes for the existence of mode 14h.

If mode 14h is supported in BIOS, issue the appropriate Video BIOS Set Mode call to put the subsystem into 132-column text mode.

If Video BIOS Mode 14h is not supported in BIOS, the following sequence of operations puts the subsystem into 132-column text mode:

1. If necessary, put the XGA subsystem into VGA mode.
2. Write data to registers in the following sequence:

Value (hex)	Oper	XGA Reg	VGA 3D4/5	VGA 3C4/5	Other VGA	Comments
15	=	50				Prepare CRT controller for reset
14	=	50				Reset CRT controller
04	=	54				Select VGA clock

Figure 3-181. 132-Column Text Mode First Write Sequence

3. Set the number of lines in VGA mode using INT 10h Ah=12h (200, 350, or 400).
4. Set VGA mode 3 using INT 10h Ax=0003h. The 132-column text mode is a variation of the VGA text mode. The following table gives the variations from the standard mode.

5. Write data to registers in the following sequence:

Value (hex)	Oper	XGA Reg	VGA 3D4/5	VGA 3C4/5	Other VGA	Comments
01	|=	50				) Prepare CRT controller for reset
FD	&=	50				)
FC	&=	50				Reset CRT controller
03	=	21x0				132-column text mode
01	=	54				132-column clock frequency select
80	=	70				Select internal 132-column clock
EF	&=	50				Disable video extension
7F	&=		11			Enable VGA CRT controller reg update
A4	=		0			)
83	=		1			)
84	=		2			)
83	=		3			)
90	=		4			) Variations on VGA CRT
80	=		5			) controller syncs
A3	=	1A				)
00	=	1B				)
00	=	1C				)
00	=	1E				)
42	=		13			)
80	|=		11			Disable VGA CRT controller reg update
03	|=	50				Remove CRT controller reset
01	|=			01		8 bit characters
**	INP				3DA	Read sets attribute controller flip flop
13	=				3C0	) Sets attribute controller
00	=				3C0	) Registers hex 13 to 00
20	=				3C0	Restore palette

|=	Logical OR to modify register contents
&=	Logical AND to modify register contents
=	Write value to register
INP	Read value from register

Figure 3-182. 132-Column Text Mode Second Write Sequence

6. Write hex 84 to 40:4A in BIOS data area to force Video BIOS recognition of 132-column text mode.

The coded text buffer is now 132 columns wide. The mode can now be programmed like any other VGA text mode, with a coded text buffer located at hex B8000 in system address space.

If it is necessary to invoke a mode change using Video BIOS (INT 10h) while in 132-column text mode (for example, to vary the number of lines), follow step 1 on page 3-185 through step 6.

System Video Memory Apertures

There are three possible apertures in the physical address space of the system. If present, any of them can be used by the system processor to access directly the packed PEL display buffer mapped into system memory. The XGA coprocessor may make the use of an aperture unnecessary.

The precise location of each aperture, and whether it is enabled, is determined by decoding the XGA subsystem POS data, as described in "Locating the XGA Subsystem" on page 3-171.

64KB System Video Memory Aperture: This aperture is at hex A0000 or B0000 in physical address space. The 64KB aperture is insufficient to access the entire subsystem display buffer. Therefore, the aperture position over the display buffer is controlled by using the Aperture Index register.

This is the only aperture in i86 real mode address space.

Other video adapters, such as another adapter or subsystem in either VGA or Extended Graphics mode, may contend for the use of this aperture. Only one video subsystem can have this aperture enabled at a time. If there is no contention for the hex A0000 or B0000 address spaces, this aperture is the only one that can be enabled by the application.

1MB System Video Memory Aperture: The base address of this aperture may be located on any 1MB boundary from 1MB to 15MB, or it may be disabled. The aperture address is determined by the system configuration. To determine its position, and whether it is enabled, decode the POS data as described in "Locating the XGA Subsystem" on page 3-171.

In systems with multiple XGA subsystems, each one may have its own aperture. Depending on the hardware configuration, it is possible for some, but not all coexisting XGA subsystems to have their 1MB system video memory apertures enabled.

This aperture is large enough to access the entire video memory without using the Aperture Index register to move the aperture. The Aperture Index register must be set to 0 when using this aperture.

This aperture is easily accessible only in protect mode environments. The operating system must provide addressability to the address range occupied by the aperture. Some operating systems attempt to restrict such addressability to protect device drivers or kernel device drivers. A small kernel device driver may need to be written to provide addressability. For example, in a 16-bit segmented system such as the OS/2 version 1.3 operating system, the following steps may be necessary to build global descriptor table (GDT) addressability to an aperture:

1. Allocate a GDT selector.
2. Modify the GDT entry directly to alter the permission bits to allow user mode (ring 3) access.
3. Alter the GDT segment length to be a 1MB segment. The entire 1MB video memory display buffers can then be accessed as a single segment.

Check that the aperture is enabled before assuming its existence. If the aperture is disabled, it cannot be enabled by the application. The application should then try to use the 4MB aperture.

4MB System Video Memory Aperture: The base address of this aperture may be located on any 4 megabyte boundary at or above 16MB, or it may be disabled. The aperture address is determined by the system configuration. To determine its position, and whether it is enabled, decode the POS data as described in "Locating the XGA Subsystem" on page 3-171.

In systems with multiple XGA subsystems, each one may have its own aperture.

This aperture is not available in 16-bit systems based on the 80386SX. This aperture does not exist when the XGA subsystem adapter is plugged into a 16-bit (short) slot on a 32-bit system.

Check that the aperture is enabled before assuming its existence. Also, check the Auto-Configuration register, as described in "Auto-Configuration Register (Index 04)" on page 3-43, to determine the bus width.

While this aperture is present when the XGA subsystem is plugged into a 32-bit slot on a 32-bit system, it may not be easily accessible in real-mode DOS or 16-bit protect-mode operating systems.

Physical Addressability to System Memory

The XGA subsystem coprocessor can operate as a Micro Channel busmaster. As a busmaster, the coprocessor is capable of bit map operations on bit maps up to 4KB by 4KB PELs anywhere in system address space, including video memory. A PxBlt operation can be defined as a function of four separate bit maps:

```
D' = f(S,D,P,M)
```

That is, the modified destination PEL (*D'*) is a function of the source (*S*), the current destination PEL (*D*), the pattern (*P*), and the mask (*M*). These bit maps can be anywhere in memory. The XGA coprocessor handles all bit maps alike. No special handling of a bit map in video memory is required.

This flexibility is very powerful, but requires support from the operating system to fully realize the benefits.

Busmastership is on i386 physical address space, while applications run on the system processor in virtual or linear address space. The system processor automatically converts such addresses to physical addresses internally through the page tables or segment descriptor tables. An adapter, such as the XGA coprocessor, has no physical access to the segment descriptors or the page tables. To use busmastership, the application (or its device drivers) must provide the XGA coprocessor with the physical address of all the bit maps on which it requires the XGA coprocessor to operate. Methods for providing the XGA coprocessor with physical addressability to all such resources, and the tasks necessary, vary according to the operating system and the mode of the system processor.

Real-Mode DOS Environments

The real-mode DOS environment is the simplest and easiest in terms of memory management. The application is limited to 640KB of real-mode DOS memory. Virtual-to-linear memory address conversion is done by means of a simple *shift left 4 and add* operation, and the nature of the real-mode DOS environment is that linear addresses are identical to physical addresses.

In the multiple virtual DOS machine (MVDM) environment, however, linear addresses are no longer identical to physical addresses, and a DOS application or device driver may not necessarily work correctly in an MVDM environment.

In most cases, the virtualization display driver of the MVDM hypervisor will cope with this, but applications must be tested in individual MVDM environments before full, real-mode DOS compatibility can be claimed.

Extended Memory: A DOS application can allocate large areas of extended memory as working bit maps for the application. It is unnecessary to have system processor addressability to such bit maps. The XGA coprocessor can do all the necessary accesses, and extended memory is ideal for this purpose.

The techniques required to allocate and use extended memory in a DOS application are not covered here.

LIM EMS Managers: The most common memory management technique that gives extra memory in the DOS environment is the Lotus-Intel-Microsoft Expanded Memory Services Manager. These memory managers implement the LIM 4.0 specification for a software interrupt driven memory management interface using software interrupt 67h. On 80386 and above processors, memory is physically allocated as extended memory, and the LIM EMS manager maps this into expanded memory using the 80386 page tables.

The drawback to this technique is that a simple shift left 4 and add operation yields the linear, but not the physical address of the LIM frame. To determine the physical address, it is necessary to call the Operating System DMA services interface of the LIM EMS driver to convert linear addresses to physical addresses. This interface, based on Software Interrupt 4Bh, is described in the *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference*.

This interface is of recent origin, and early LIM drivers may not have implemented it. There are two choices for the application:

- Do not locate resources in LIM memory on which the XGA coprocessor is requested to operate.
- Specify a dependency in the application documentation on LIM EMS drivers that have implemented this interface.

32-Bit DOS Extended Environments: In this mode, exploitation of the power of the XGA coprocessor is easiest. The application can allocate large memory bit maps without accounting for the behavior of a memory manager that might change the location of the memory. Calculation of physical addresses is easily accomplished without the system overheads of full-blown protect mode operating systems.

Access to the XGA system video memory aperture and coprocessor register address space can be accomplished easily.

Multiple Virtual DOS Machine Environments: In this mode, multiple DOS applications can run concurrently (even windowed on the same screen) in virtual DOS machines (VDMs) and each application appears internally to be running in the bottom 1MB of physical address space.

Full compatibility with real-mode DOS for a busmaster, such as the XGA coprocessor, is only provided if each DOS application using the XGA subsystem in Extended Graphics mode is locked in the bottom 1MB of physical address space. Because this is impractical, the virtualization display driver (VDD) of the MVDM hypervisor must include specific support functions to support VDMs running Extended Graphics mode applications.

One technique is described here, although there may be others that are equally effective.

When a VDM that is running an XGA Extended Graphics mode DOS application switches to the foreground, the VDD locks the entire 640KB of linear address space in the VDM without making the memory contiguous. The VDD then uses the page directory entry (PDE) of the foreground VDM to provide physical addressability to the noncontiguous linear address space. The virtual address capability of the XGA coprocessor can then be used by giving the XGA coprocessor direct DMA access to the page tables of the VDM. Because the entire 640KB DOS region is locked (except for LIM which will be discussed below), a DOS application will not normally supply linear addresses outside that range.

The Extended Graphics mode DOS application must not modify the XGA coprocessor Page Directory Base Address after it is set by the VDD when switching the VDM to the foreground. Application updates to this field can be prevented by placing the XGA coprocessor into user mode.

The DOS application may locate a resource, such as a font definition, in LIM memory and give the XGA coprocessor the linear address of the LIM frame, rather than the underlying address. This is normally handled in real-mode DOS by calling the *Operating System DMA Services* interface of the LIM EMS driver to convert linear addresses to physical. In the MVDM environment, the XGA coprocessor is in VM mode, and the linear address of the LIM frame is required, rather than the physical address. The VDD can monitor the LIM software interrupt (Int 67h), and ensure that any LIM *logical 16K pages* currently mapped into the LIM frames or windows of the VDM are locked. The page tables of the VDM will then naturally reflect the correct physical addresses for the LIM pages at the linear address of the LIM frame. Calls to the *Operating System DMA Services* interface must also be filtered out.

Protect Mode 16-Bit Segmented Environment: An application written for this environment has a range of limitations imposed by the operating system.

64KB Segment Limit: No memory object in this environment can be larger than 64KB, unless allocated by a kernel device driver on initialization.

The application cannot assume that two adjacent segments are located adjacently in physical address space.

Segment Motion: Segments may be moved in physical system memory at any time. Segments may even be swapped out to disk when memory is over committed.

All segments must be locked before the physical address is established.

Consideration must be given to the overall impact on system performance of locking large areas of memory. Locking increases the minimum physical memory configuration required to run the application.

System Overheads: Applications generally run at a low privilege level and video device drivers must be easily and frequently accessible by the application without large system overheads.

Applications using the XGA coprocessor need to make use of the memory management services of the operating system. These services (used for locking segments and determining the physical address of segments) are typically restricted to device drivers operating at high privilege levels.

The system overhead in reaching these services in such operating systems can be so high that it makes the writing of high performance applications difficult.

Access to XGA Registers and System Memory Apertures: Ingenuity is required to provide addressability to the I/O and memory space of the XGA subsystem. A technique for this is described in "System Video Memory Apertures" on page 3-187.

Following is a suggested design for an application in this environment. This technique minimizes kernel or system overheads.

Use a kernel or ring 0 .SYS device driver to permanently allocate a range of physical memory (typically 128K) as kernel work space (KWS). The device driver can then generate a GDT selector to the KWS that is valid in user mode at ring 3. Both the virtual and physical addresses of the KWS are passed back to the application in user mode. The kernel device driver also provides user mode addressability to the register address space of the XGA coprocessor.

The device driver or application can then operate totally in user mode, passing resources (for example, bit maps or patterns) by system processor block moves into the KWS. The application can then use the busmastership capability of the XGA coprocessor to access the resources in the KWS without suffering the system overheads of switching into kernel mode again. Bit maps being transferred to or from the adapter can be double buffered through the KWS to overlap system processor and XGA coprocessor operations on large operations.

Paged Virtual Memory (virtual memory) Environments: This environment shares many constraints with the 16-bit segmented environment. The main difference is that the unit of granularity of memory objects has dropped from 64KB to 4KB; the virtual memory support in the XGA coprocessor is intended to support this environment.

4KB Discontiguous Pages: In this environment, memory is allocated to applications in 4KB pages. The system memory manager controls paging and can swap pages in and out of physical memory transparently to the application. The application can make no assumptions about the relationship between adjacent pages.

There are memory management calls available to the kernel or ring 0 device driver that let the device driver build a table containing the physical addresses of all the component pages of a large bit map. As with 16-bit segmented environments, described in "Protect Mode 16-Bit Segmented Environment" on page 3-192, the overhead of the transition to kernel mode makes such calls expensive. It is, however, possible to build such a table and to operate the XGA coprocessor in virtual memory mode. The overall impact on system performance and minimum physical memory configurations should be considered. A bit map in this case could theoretically be 4Kx4Kx8 bits per PEL, which is a total of 16MB of locked physical memory.

It is possible to use the XGA coprocessor to interrupt (to indicate a page fault). However, this interrupt is a normal shared-adapter interrupt rather than a i386 page fault interrupt, and is handled at a lower priority. Most operating systems do not allow device drivers to call the memory management services to request the faulting pages.

Page Table Coherency: It may appear that the XGA coprocessor can operate off the system page tables because the XGA coprocessor uses i386-like page tables. Unfortunately, a typical virtual memory operating system uses one set of page tables per task. In a multitasking environment, only the currently executing task page tables remain coherent, while background task page tables become outdated or incoherent.

This implies that the XGA coprocessor can be operating on a set of page tables belonging to a background task. It cannot be assumed that the page table remains coherent, unless the component pages have been locked by a call to the system memory management interface by a kernel device driver.

System Overheads: The overheads associated in switching from the application privilege level to the kernel level have been described in *System Overheads* in "Protect Mode 16-Bit Segmented Environment" on page 3-192.

Access to XGA Registers and System Memory Apertures: It is necessary to provide addressability to these XGA subsystem I/O spaces. Call the operating system memory management services to map these ranges of physical system memory into the application task address space.

Suggested Design Model: The optimum design model is one that minimizes kernel overhead. A model similar to that suggested in "Protect Mode 16-Bit Segmented Environment" on page 3-192 is appropriate for this environment.

Video Memory Addressability in Virtual Memory Mode: "Video Memory Location in Coprocessor Address Space" on page 3-174 has a description of how the XGA coprocessor differentiates video memory from system memory. When operating the XGA subsystem in VM mode, this differentiation is done after page table translation on physical address space. All addresses passed to the XGA coprocessor by the application or device driver are in linear address space, before page table translation. When the application or device driver is building VM addressability to system memory bit maps for the XGA subsystem, it must also map local video memory into the page table structure at the correct location in physical address space to allow the XGA coprocessor to differentiate video memory from system memory.

System Memory Access Limitation: The XGA subsystem can be plugged into any 16- or 32-bit slot in any i386SX, i386DX, or i486 system. In a 16-bit slot, the address range is limited because there are only 24 address lines on 16-bit slots. The range of physical addressability to system memory using busmastership is limited to 24-bit physical address space (or 16MB) when the subsystem occupies a 16-bit slot.

Systems based on the i386SX are 16-bit throughout. The limit of addressability of the system processor is 16MB.

There are constraints when:

- A 32-bit system is based on the i386DX or i486
- There is more than 16MB of physical memory installed
- The XGA subsystem is plugged into a 16-bit slot.

  The XGA coprocessor cannot access memory located above the 16MB line in physical address space. To determine if the XGA subsystem is in a 16-bit slot, examine the Auto-Configuration register, as described in "Auto-Configuration Register (Index 04)" on page 3-43. The application must ensure (with operating system assistance if necessary) that all memory bit maps on which the XGA processor is asked to operate are located below the 16MB line in physical address space.

  The alternative is for the application to specify that the XGA subsystem is always plugged into 32-bit slots on 32-bit systems.

Upward Compatibility

Upward compatibility problems can be minimized by sensible programming practices, and the following precautions.

XGA Subsystem POS ID Allocations

The following POS IDs have been preallocated to the XGA subsystem and follow-on XGA register compatible subsystems:

- hex 8FD8
- hex 8FD9
- hex 8FDA
- hex 8FDB.

Check for these POS IDs when determining the existence and location of the XGA subsystem in the system.

General Register Usage

To avoid conflicts with possible future changes in the use of registers or register fields, applications must comply with the Register Usage Guidelines at the start of the various register definition sections.

Video BIOS Mode 14h

Video BIOS mode 14h has been reserved to support the 132-column text mode. Applications should plan to use BIOS support for this mode as it becomes available. They must query Video BIOS for the existence of the mode. Given a positive response, the Video BIOS INT 10h Set mode must be used for mode setting. Only in the absence of INT 10h Video BIOS support should the direct mode setting procedure described in this section be used.

PS/2 Video Memory Apertures

As described in "System Video Memory Apertures" on page 3-187, not all of the apertures may exist. However, the A000 or B000 aperture can always be enabled under application control, depending on which is available. For maximum flexibility, applications should not use the apertures. If this is not possible, the following considerations apply:

- If the XGA subsystem is not plugged into a 16-bit slot on a 32-bit system, at least one of the two potential protect mode apertures should be available.
- Inform the application user that the XGA subsystem must be installed in a 32-bit slot on a 32-bit system.

Programming the XGA Subsystem in Extended Graphics Mode

This section describes using Extended Graphics functions of the XGA coprocessor.

XGA Coprocessor PEL Interface Registers

Extended graphics functions are graphics update operations involving up to four PEL maps. A PEL map is defined by five registers:

- PEL Map Index register
- PEL Map n Base Pointer register
- PEL Map n Width register
- PEL Map n Height register
- PEL Map n Format register.

PEL Map Index Register: This register has an offset of hex 12. The PEL Map Index register defines which of the four possible maps is to be defined. The encoding of this 4-bit register is as follows:

Mask map	hex 0
PEL map A	hex 1
PEL map B	hex 2
PEL map C	hex 3

For example, to use PEL map A:

```
WRITE 01h to copr_regs offset 12h.
```

PEL Map Base Address Register: This register has an offset of hex 14. The PEL Map Base Pointer register defines the byte address in memory of the start of the PEL map. It is a 32-bit address register and can therefore address up to 4096MB of memory. A PEL map can be defined to be in the XGA video memory or in system memory.

As described in “Video Memory Location in Coprocessor Address Space” on page 3-174, to define a PEL map as being in XGA video memory, the address put in this register must be in the following range:

```
Video Memory Base Address ↔ (Video
Memory Base Address + Video Memory size)
```

If the PEL map is in system memory and the Micro Channel interface is a 16-bit interface (for example, if the XGA adapter is installed in a 16-bit slot), the address of the map must be below 16MB.

PEL Map Width Register: This register has an offset of hex 18. The PEL map width is measured in PELs and is defined as one less than the required width.

For example, to set the width of a PEL map to 640 PELs:

```
WRITE 027Fh to copr_regs offset 18h
```

To set the width of a PEL map to 1024 PELs:

```
WRITE 03FFh to copr_regs offset 18h
```

PEL Map Height Register: This register has an offset of hex 20. The PEL map height is measured in PELs and is defined as one less than the required height.

For example, to set the height of a PEL map to 480 PELs:

```
WRITE 01DFh to copr_regs offset 20h
```

To set the height of a PEL map to 768 PELs:

```
WRITE 02FFh to copr_regs offset 20h
```

PEL Map Format Register: This register has an offset of hex 1C. This register specifies the bits per PEL of the PEL map. The encoding of the register is as follows:

1 bit/PEL Intel format	hex 00
2 bits/PEL Intel format	hex 01
4 bits/PEL Intel format	hex 02
8 bits/PEL Intel format	hex 03
1 bit/PEL Motorola format	hex 08
2 bits/PEL Motorola format	hex 09
4 bits/PEL Motorola format	hex 0A
8 bits/PEL Motorola format	hex 0B

For example, for an 8-bit/PEL Motorola format PEL map:

```
WRITE 0Bh to copr_regs offset 1Ch
```

The relationship between Intel and Motorola format PEL maps is discussed in "Video Memory Format" on page 3-16 and "Motorola and Intel Formats" on page 3-219.

All four PEL maps (A, B, C, and mask) can be initialized in this manner to be ready for later use. Maps A, B, and C can be used interchangeably as the source, destination, or pattern in all subsequent PEL operations.

Other Registers: For simple operations, the PEL Interface Control register must be cleared.

For example:

```
WRITE 00h  to copr_regs offset 11h
```

For simple operations, the Destination Color Condition Compare register must be set so that it has no effect on the operation.

For example:

```
WRITE 04h  to copr_regs offset 4Ah
```

To allow all planes of a PEL map to be updated, the PEL bit mask must be turned on. That is, set all bits to 1 that are required for the PEL size selected.

For example, for 8-bits-per-PEL:

```
WRITE 00FFh to copr_regs offset 50h
```

For simple operations, the carry chain mask must be turned on. That is, set all bits to 1 that are required for the PEL size selected.

For example, for 8-bits-per-PEL:

```
WRITE FFh to copr_regs offset 54h
```

Using the Coprocessor to Perform a PEL Blit (PxBlt)

This section describes the actions necessary to use the XGA coprocessor to perform a simple PxBlt.

Various types of PxBlt can be performed. This example is for a PxBlt into video memory using the Foreground Color register as the source data. The result is a solid rectangle drawn into the display PEL map. The example PxBlt has the following characteristics:

- Foreground color of hex 05
- 100 PELs wide and 60 PELs deep
- Positioned at screen coordinates X=200 and Y=150.

The following table lists the values that must be written to the coprocessor registers. Each value is explained following the table, along with information on the other forms of PxBlt available.

Value (hex)	Coprocessor Registers Offset (hex)
03	48
05	58
0063	60
003B	62
00C8	78
0096	7A
08118000	7C

Figure 3-183. Coprocessor Register Write Values

Mixes and Colors: Before a coprocessor operation can be performed, the background and foreground mixes have to be set. Mixes are logical or arithmetic functions performed on the source and destination data when performing a coprocessor operation. The mix functions available are as follows:

Code	Function
0	Zeros
1	Source AND Destination
2	Source AND NOT Destination
3	Source
4	NOT Source AND Destination
5	Destination
6	Source XOR Destination
7	Source OR Destination
8	NOT Source AND NOT Destination
9	Source XOR NOT Destination
A	NOT Destination
B	Source OR NOT Destination
C	NOT Source
D	NOT Source OR Destination
E	NOT Source OR NOT Destination
F	Ones
10	Maximum
11	Minimum
12	Add with Saturate
13	Subtract (Destination – Source) with Saturate
14	Subtract (Source – Destination) with Saturate
15	Average

Figure 3-184. Background and Foreground Mixes and Colors

Foreground and Background Mix Registers: The mixes to be applied to foreground and background PELs are specified in these two registers. The contents of the pattern map determine the PELs for foreground and background. In this example, the PxBlt is solid and contains only foreground PELs. The Foreground Mix register must be set to Source to give an understandable result on the screen.

For the example:

```
WRITE 03h  to copr_regs offset 48h
```

Foreground and Background Color Registers: The colors to be used for foreground and background PELs are specified in these two registers. In this example, the PxBlt is solid and only the Foreground Color register needs to be set up.

For the example:

```
WRITE 05h to copr_regs offset 58h
```

Other forms of PxBlt (for example, video memory to video memory) from a source map into a destination map do not use these color registers.

PxBlt Dimensions: The Operation Dimension 1 register must be loaded with the Width of PxBlt to be performed. The value loaded into the register must be one PEL less than the required width (in PELs).

For example, for a 100-PEL wide Pxblt:

```
WRITE 0063h to copr_regs offset 60h
```

The Operation Dimension 2 register must be loaded with the Height of PxBlt to be performed. The value loaded into the register must be one PEL less than the required height (in PELs).

For example, for a 60-PEL high Pxblt:

```
WRITE 003Bh to copr_regs offset 62h
```

PEL Map, Source, and Destination Registers

Source Map X and Y Registers: The source map is initialized as detailed in “Mixes and Colors” on page 3-202. Within the source map, two registers exist that contain the X and Y offset positions of the start of the source data for a PxBlt. These registers are used when performing a PxBlt using a source map. In this example, these registers are unused.

Destination Map X and Y Registers: The destination map is initialized as detailed in “Mixes and Colors” on page 3-202. Within the destination map, two registers exist that contain the X and Y offset positions of the start of the PxBlt.

For the example, to position the PxBlt at X=200 and Y=150 in the destination map:

```
WRITE 00C8h to copr_regs offset 78h (Destination Map X position)

WRITE 0096h to copr_regs offset 7Ah (Destination Map Y position)
```

Pattern Map X and Y Registers: The pattern map is initialized as detailed in “Mixes and Colors” on page 3-202. Two registers exist that contain the X and Y offset positions, within the pattern map, of the start of the pattern data for a PxBlt. These registers are used when performing a PxBlt using a pattern map.

In this example these registers are unused.

Mask Map Origin X and Y Offset Registers: The mask map is initialized as detailed in the previous chapter. Two registers exist that contain the X and Y offset positions of the start of the mask map relative to the top left corner of the destination map. These registers are used when performing a PxBlt using a mask map.

In this example these registers are unused.

PEL Operations Register: This is a 32-bit register that defines the operation the coprocessor performs.

31	30	29	28	27			24	23			20	19			16	15			12	11			8	7	6	5	4	3	2		0
																				X	X	X	X					X			
1		2		3				4				5				6								7		8			9		

Figure 3-185. Bit Layout PEL Operations Register

The definition of the fields at the bottom of Figure 3-185 are:

1. Background Source
2. Foreground Source
3. Step Function
4. Source PEL Map
5. Destination PEL Map
6. Pattern PEL Map
7. Mask PEL Map
8. Drawing Mode
9. Direction Octant.

These fields, described in sequence, are required to assemble the PEL Operations register:

Background Source: These bits determine the origin of the background source PELs when an operation is performed.

The encoding for these bits is as follows:

Background Color	Binary 00 (for example, for a fixed register value to video memory PxBlt).
Source PEL Map	Binary 10 (for example, for a video memory to video memory PxBlt).

For this example, there is no background color, and the field is ignored.

```
Background Source = Binary 00
```

Foreground Source: These bits determine the origin of the foreground source PELs when an operation is performed.

The encoding for these bits is as follows:

Foreground Color	Binary 00 (for example, for a fixed register value to video memory PxBlt).
Source PEL Map	Binary 10 (for example, for a video memory to video memory PxBlt).

For this example there is a solid foreground color:

```
Foreground Source = Binary 00
```

Step Function: These bits define the type of operation that the coprocessor is required to do.

The encoding for these bits is as follows:

Draw and Step Read	Binary 0010
Line Draw Read	Binary 0011
Draw and Step Write	Binary 0100
Line Draw Write	Binary 0101
PxBlt	Binary 1000
Inverting PxBlt	Binary 1001
Area Fill PxBlt	Binary 1010

For this example:

```
Step Function = binary 1000
```

Source PEL Map: These bits define the PEL map used as the source map in the operation. This enables different maps to be setup in advance and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011

For this example, the contents of this field are ignored but they must not be a reserved value.

```
Source PEL Map = Binary 0001
```

Destination PEL Map: These bits define the PEL map used as the destination map in the operation. This enables different maps to be setup in advance and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011

For this example:

```
Destination PEL Map = Binary 0001
```

Pattern PEL Map: These bits define the PEL map used as the pattern map in the operation. This enables different maps to be setup in advance and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011
Foreground (fixed)	Binary 1000
Generated from Source	Binary 1001

For this example:

```
Pattern PEL Map = binary 1000
```

Mask PEL Map: These bits define whether the mask map is used in the operation.

The encoding for these bits is as follows:

Mask Map Disabled	Binary 00
Mask Map Boundary Enabled	Binary 01
Mask Map Enabled	Binary 10

For this example:

```
Mask PEL Map = Binary 00
```

Drawing Mode: These bits concern line drawing only and are discussed later. They are ignored during a PxBlt.

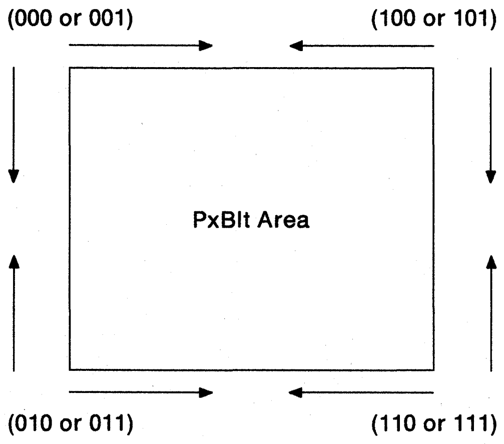
For this example:

```
Drawing Mode = Binary 00
```

Direction Octant: These bits, when concerned with PxBlts, determine the direction in which the PxBlt is drawn.

The encoding for these bits is as follows:

Binary 000 or 001	Start at top left-hand corner of area increasing right and down.
Binary 100 or 101	Start at top right-hand corner of area increasing left and down.
Binary 010 or 011	Start at bottom left-hand corner of area increasing right and up.
Binary 110 or 111	Start at bottom right-hand corner of area increasing left and up.



Note: Numbers are binary.

Figure 3-186. Operation Direction Diagram

These bits are normally set to binary 000, but other values are necessary to avoid PEL corruption when source and destination rectangles overlap.

For this example the PxBlt is in top left corner:

```
Direction Octant = Binary 000
```

Conclusion: Putting all these together for the PxBlt, the PEL Operations register must be set as:

31	30	29	28	27			24	23			20	19			16	15			12	11			8	7	6	5	4	3	2		0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	X	X	X	X	0	0	0	0	X	0	0	0

Figure 3-187. Definition for PEL Operations Register (Example)

For this example:

```
WRITE 08118000h to copr_regs offset 7Ch
```

Using the Coprocessor to Perform a Bresenham Line Draw

The steps required to draw a line of palette color hex 05 from (20,15) to (80,35), are summarized in the following table. The sections that follow explain each value and provide information on the other line drawing options available.

Value	Coprocessor Registers Offset (hex)
Hex 03	48
Hex 05	58
Decimal − 20	20
Decimal 40	24
Decimal − 80	28
Decimal 60	60
Decimal 20	78
Decimal 15	7A
Hex 05118000	7C

Figure 3-188. Palette Color Line Draw Steps

Mixes and Colors: Before a coprocessor operation is performed, the background and foreground mixes have to be set. Mixes are logical or arithmetic functions performed on the source and destination data when performing a coprocessor operation. The mix functions available are as follows:

Code	Function
0	Zeros
1	Source AND Destination
2	Source AND NOT Destination
3	Source
4	NOT Source AND Destination
5	Destination
6	Source XOR Destination
7	Source OR Destination
8	NOT Source AND NOT Destination
9	Source XOR NOT Destination
A	NOT Destination
B	Source OR NOT Destination
C	NOT Source
D	NOT Source OR Destination
E	NOT Source OR NOT Destination
F	Ones
10	Maximum
11	Minimum
12	Add with Saturate
13	Subtract (Destination - Source) with Saturate
14	Subtract (Source - Destination) with Saturate
15	Average

Figure 3-189. Background and Foreground Mixes and Colors

Foreground and Background Mix Registers: The Foreground Mix and Background Mix registers allow a mix (as detailed in the table) to be specified. These registers are discussed in the previous example, “Using the Coprocessor to Perform a PEL Blit (PxBlt)” on page 3-201.

For this example, the Foreground Mix register must be loaded with Source. The Background Mix register is not used in this example.

For the example with the Foreground Mix register:

```
WRITE 03h to copr_regs offset 48h
```

Foreground and Background Color Registers: The Foreground Color register must be set to the color required for the line.

For this example with the Foreground Color register:

```
WRITE 05h to copr_regs offset 58h
```

Bresenham Line Draw: The algorithm used to perform the line draw function on the XGA is the Bresenham Line Draw algorithm. This operates with all parameters normalized to the first octant (octant 0).

The first task is to calculate deltaX and deltaY (see the following figure).

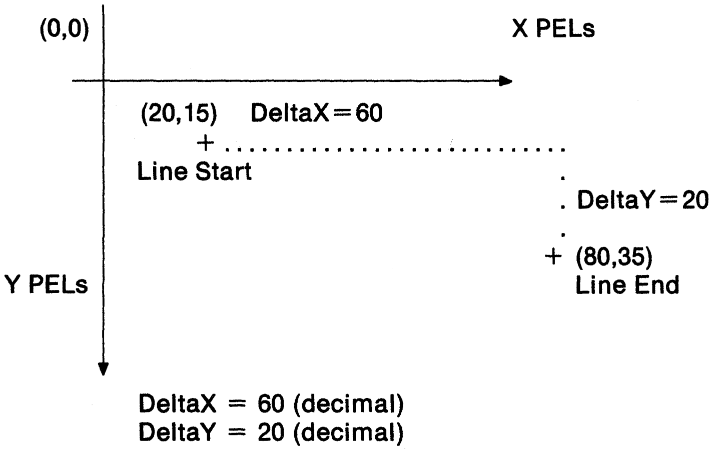


Figure 3-190. Line Draw Example in Octant 0

A line in the first octant has deltaX greater than deltaY, with both deltaX and deltaY positive, and deltaX greater than deltaY. If a line is to be drawn in another octant, the octant information is specified in the octant bits of the PEL Operation register. The line is drawn as if it were in the first octant.

To normalize a line to the first octant, follow these rules:

- If deltaX is −ve , set DX in octant bits of the PEL Operation register and make deltaX +ve.
- If deltaY is −ve , set DY in octant bits of the PEL Operation register and make deltaY +ve.
- If deltaY ≥ deltaX , set DZ in octant bits of the PEL Operation register and exchange deltaX and deltaY.

The terms deltaX and deltaY are the lengths of the line after it has been normalized to octant 0. The algorithm requires several parameters to be calculated. These are:

Bresenham Error Term Register: Bresenham Error Term
$E = (2 \times \text{deltaY}) - \text{deltaX}$

For this example:

```
WRITE -20 decimal (FFECh) copr_regs offset 20h
```

Bresenham Constant K1 Register: Bresenham Constant
$K1 = 2 \times \text{deltaY}$

For this example:

```
WRITE +40 decimal (0028h) copr_regs offset 24h
```

Bresenham Constant K2 Register: Bresenham Constant
$K2 = 2 \times (\text{deltaY} - \text{deltaX})$

For this example:

```
WRITE -80 decimal (FFB0h) copr_regs offset 28h
```

Operation Dimension Registers: The Operation Dimension 1 register should be loaded with deltaX after normalization. Because a value of 0 results in a line length of 1 PEL, deltaX (calculated in Figure 3-190 on page 3-211) equals the number to be drawn minus 1.

For this example:

```
WRITE +60 decimal (003Ch) to copr_regs offset 60h
```

The Operation Dimension 2 register is not used for line draw.

PEL Map Source and Destination

Source Map X and Y Registers: The source map is initialized as described in "XGA Coprocessor PEL Interface Registers" on page 3-197. Two registers exist that contain the X and Y offset positions within the source map of the start of the source data for a PxBlt. These registers are used for drawing a line using a source map. In this example, these registers are unused.

Destination Map X and Y Registers: The destination map is initialized as described in "XGA Coprocessor PEL Interface Registers" on page 3-197. Two registers exist that contain the X and Y offset positions within the destination map of the start of the line.

In this example with destination map X and Y positions:

```
WRITE 0014h to copr_regs
offset 78h

WRITE 000Fh to copr_regs
offset 7Ah
```

Pattern Map X and Y Registers: The pattern map is initialized as described in "XGA Coprocessor PEL Interface Registers" on page 3-197. Two registers exist that contain the X and Y offset positions within the pattern map of the start of the pattern data for a line. These registers are used when drawing a line using a pattern map. In this example, these registers are unused.

Mask Map Origin X and Y Offset Registers: The mask map is initialized as described in "XGA Coprocessor PEL Interface Registers" on page 3-197. Two registers exist that contain the X and Y offset positions of the start of the mask map relative to the top left corner of the destination map. These registers are used when drawing a line using a mask map. In this example, these registers are unused.

PEL Operations Register: This is a 32-bit register that defines the operation that the coprocessor performs.

31 30	29 28	27 24	23 20	19 16	15 12	11 8	7 6	5 4	3	2 0
						X X X X			X	
1	2	3	4	5	6		7	8		9

Figure 3-191. Bit Layout PEL Operations Register

The bits 0−31 are shown on the top of Figure 3-191; Fields 1 through 9 are shown on the bottom. The definition of these fields is:

1. Background Source
2. Foreground Source
3. Step Function
4. Source PEL Map
5. Destination PEL Map
6. Pattern PEL Map
7. Mask PEL Map
8. Drawing Mode
9. Direction Octant.

These fields, described in sequence, are required to assemble the contents of the PEL Operations register:

Background Source: These bits determine the origin of the background source PELs when an operation is performed.

The encoding for these bits is as follows:

Background Color	Binary 00 (for example, for a fixed pattern line draw using a fixed register value)
Source PEL Map	Binary 10 (for example, for a variable color data pattern held in video memory to video memory draw).

In this example, the contents of this field are ignored because the line is solid and there are no background PELs:

```
Background Source = Binary 00
```

Foreground Source: These bits determine the origin of the foreground source PELs when an operation is performed.

The encoding for these bits is as follows:

Foreground Color	Binary 00 (for example, for a fixed pattern line draw using a fixed register value).
Source PEL Map	Binary 10 (for example, for a variable color data pattern held in video memory to video memory draw).

For this example:

```
Foreground Source = Binary 00  (Solid Foreground Color)
```

Step Function: These bits define the type of operation that the coprocessor is required to do.

Draw and Step Read	Binary 0010
Line Draw Read	Binary 0011
Draw and Step Write	Binary 0100
Line Draw Write	Binary 0101
PxBlt	Binary 1000
Inverting PxBlt	Binary 1001
Area Fill PxBlt	Binary 1010

For this example:

```
Step Function = binary 0101
```

Source PEL Map: These bits define the PEL map used as the source map in the operation. This enables different maps to be setup in advance, and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011

In this example the contents of this field are ignored, but they must not be a reserved value:

```
Source PEL Map = binary 0001
```

Destination PEL Map: These bits define the PEL map used as the destination map in the operation. This enables different maps to be setup in advance and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011

For this example with PEL map A:

```
Destination PEL Map = binary 0001
```

Pattern PEL Map: These bits define the PEL map used as the pattern map in the operation. This enables different maps to be setup in advance and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011
Foreground (fixed)	Binary 1000
Generated from Source	Binary 1001

For this example:

```
Pattern PEL Map = binary 1000
```

Mask PEL Map: These bits define whether mask map is used in the operation.

The encoding for these bits is as follows:

Mask Map Disabled	Binary 00
Mask Map Boundary Enabled	Binary 01
Mask Map Enabled	Binary 10

For this example:

```
Mask PEL Map = Binary 00
```

Drawing Mode: These bits determine the attributes of a line draw.

The encoding for these bits is as follows:

Draw All PELs	Binary 00
Draw First PEL Null	Binary 01
Draw Last PEL Null	Binary 11
Mask Area Boundary	Binary 11

The first three options can be used when drawing a line. The fourth option is for use when drawing the outline of a shape to be filled using the area fill capability of the hardware.

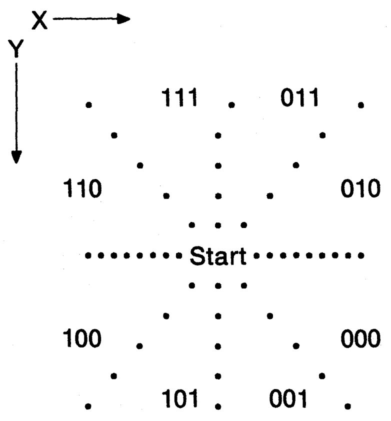
For this example for draw all PELs:

```
Drawing Mode = Binary 00
```

Direction Octant: These bits, when concerned with line draws, determine the direction in which the line is drawn.

The encoding for these bits is as follows:

Bit 2(DX)	1	if negative X direction
	0	if positive X direction
Bit 1(DY)	1	if negative Y direction
	0	if positive Y direction
Bit 0(DZ)	1	if $\lvert X \rvert \leq \lvert Y \rvert$
	0	if $\lvert X \rvert > \lvert Y \rvert$, (magnitude)



Note: Numbers are binary.

Figure 3-192. Direction Octant (Example)

For this example:

```
Direction Octant = binary 000 (X + ve, Y + ve, |X| > |Y|)
```

Conclusion: Putting all these together for the example line draw operation, the PEL Operations register must be set as:

31	30	29	28	27			24	23			20	19			16	15			12	11			8	7	6	5	4	3	2		0
0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	1	1	0	0	0	X	X	X	X	0	0	0	0	X	0	0	0

Figure 3-193. Definition for PEL Operations Register (Example)

For this example:

```
WRITE 05118000h to copr_regs offset 7Ch
```

Memory Access Modes

This register has an address of hex 21x9. The Memory Access Modes register is used to control the format of the data supplied by the system processor through a system video memory aperture. For conventional use, this register must be set to match the format of the data as seen by the system processor (Motorola or Intel), and the depth of the video memory bit map.

Through this register, the different formats available can be used to achieve useful and otherwise difficult conversions.

Motorola and Intel Formats

The internal organization of the video memory is Intel format. However, images and bit maps are traditionally stored in Motorola format. It is necessary to understand the format of the application's bit maps in system memory to get the correct results. The different formats are described in "Video Memory Format" on page 3-16.

The internal organization of video memory as Intel format can be hidden by appropriate use of the Memory Access Mode register ("Memory Access Modes") and the various coprocessor PEL map format registers.

System Processor Access: When using the system processor to read or write data directly to or from video memory through a system video memory aperture, it is necessary to specify the format of the data using the Memory Access Mode register.

XGA Coprocessor Accesses: The format of all bit maps in system memory must be specified through the PEL Map Format register. This parameter is ignored for bit maps in video memory.

Exploitation: Writing data in one format and reading it back in another is a technique that performs many useful and otherwise difficult or expensive bit map conversions.

Other Programming Considerations

Overlapping PxBlts

PEL Block Transfer (PxBlt): The coprocessor PxBlt function is used to transfer a rectangular block of PELs from the source to the destination subject to a number of modifiers. It is important to predetermine whether the source and destination rectangles overlap. If the rectangles do not overlap, the order of processing PELs is immaterial. If the rectangles do overlap, program the PxBlt direction using the Direction Octant to ensure the expected result.

Inverting PxBlt: The inverting PxBlt is intended to convert images from the traditional application format of Y increasing upwards to the traditional display hardware format of Y increasing downwards. As such a PxBlt operates from both ends towards the middle, an Inverting PxBlt involving overlapping source and target rectangles inevitably overwrites PELs. Therefore, inverting PxBlts on overlapping rectangles should be avoided, unless for special effects.

Sprite Handling

Sprite Loading: The sprite is loaded as a 64 x 64 x 2 bits per PEL (bpp) Intel format image definition. Because the sprite definition in the application is invariably held in two separate 1-bpp Motorola format bit maps, it is necessary to merge and PEL swap the sprite definition into the 2-bpp Intel format before loading the sprite.

Sprite Positioning: The position of the sprite is then controlled by two separate controls:

Sprite Start Registers
: The sprite is positioned on the display surface by specifying the position of the top left corner of the sprite definition relative to the top left corner of the visible bit map, using the Sprite Horizontal Start and Sprite Vertical Start registers.

Sprite Preset Registers
: The sprite start registers only accept positive values, and cannot be used to move the sprite partially off the display surface at the left and top edges. The Sprite Horizontal Preset and Sprite Vertical Preset registers are used to offset the start of the displayed sprite definition relative to the loaded definition.

For example, to display a 64 x 64 PEL sprite with the leftmost 32 PELs outside the left edge of the display surface, set the Sprite Horizontal Start register to 0, and the Sprite Horizontal Preset register to 32. The start position is now preset to the center of the loaded definition, giving the required effect.

The sprite preset can also be used to display sprites smaller than 64 x 64 PELs.

Waiting for Hardware Not Busy

The XGA coprocessor operates asynchronously with the system processor. Wait for the previous operation to complete before issuing the next operation. There are two ways to do this:

Polling the Busy Bit: Poll the coprocessor busy bit in the XGA Coprocessor Control register to determine whether the previous operation has completed before initiating the next operation.

Continuous polling of this bit slows down the coprocessor because it pauses in its current operation to process the read of the Coprocessor Control register.

If this method is chosen, it is advisable to code a double polling loop that checks the coprocessor busy bit, for example, once every 100 times around the loop.

The advantages of using this method are:

- Minimal overhead. For typical PxBlts used to display text, the previous PxBlt operation is almost complete before the system processor is ready to issue the next operation.
- Simplicity.

The disadvantages of using this method are:

- Frequent use delays the XGA coprocessor. This can be partially reduced by using a double-polling loop as described above.
- The processor is kept busy doing nothing, although it has to be doing nothing for a long time to exceed the interrupt response codepath.

Operation Complete Interrupt: The coprocessor can be programmed to cause an interrupt to the system processor when an operation is completed.

This interrupt is a shared level. Interrupt response time therefore depends on other interrupt handlers chained on this shared level. In protect mode operating systems in particular, the overheads and restrictions placed on interrupt handlers may make the performance of this technique prohibitive.

Advantages of using this method are:

- The XGA coprocessor is not slowed while waiting for completion.
- The system processor may be freed up for other tasks.

Disadvantages of using this method are:

- Program complexity.
- Interrupt response time gives a threshold in size of operation that is only exceeded by large PxBlt operations. The more complex the operating system, the higher the interrupt response time, and the larger the operation must be to benefit from using interrupts to notify the application of operation complete.

Destination Bit Map Width Restriction

Incorrect results can be obtained if the XGA coprocessor is used to write over the edge of a destination bit map where the edge of the bit map is not 4-byte aligned. To avoid this, use one of the following methods:

- Ensure that all destination bit maps have a base address that is on a 4-byte boundary and are an exact multiple of 4 bytes wide.

  The visible display bit map naturally complies with this restriction.
- Where bit maps are not aligned, software clip all PxBlts in advance so that the destination bit map boundary is not crossed during the PxBlt.

Line Length Restriction

The XGA coprocessor Destination X Address and Destination Y Address registers accept coordinates in the range (−2048 to 6143). This gives a guardband effect, where it is possible to write coordinates anywhere in this range, and the operation is hardware scissored to the edge of the destination bit map. The limit on bit map size for coprocessor operations is 0 to 4095.

Because the Operation Dimension 1 register only accepts values in the range (0 to 4095), it is not possible to draw a line in a single operation across the entire guardband coordinate space.

A two-stage line draw can be performed easily, since the line parameters (for example, ET, K1, K2, Destination X and Y, Pattern X) are already set up in the hardware at the end of the previous line segment. It is only necessary to update the new line length in the Operation Dimension 1 register to draw the remainder of the line.

System Register Usage

When programming the XGA subsystem, it is often necessary to maintain addressability to:

- XGA coprocessor memory mapped address space
- XGA state data segment (application dependent) containing the I/O base address, in other words the location of the XGA registers in I/O space
- The normal function dependent application data, such as parameter blocks
- Global application dependent data.

Many of the XGA registers are 32-bit registers.

To program the XGA subsystem efficiently, it is helpful to use the full i386 register set, specifically the FS and GS segment registers and the 32-bit extended data registers.

Use of the extra segment registers allows concurrent addressability to all the separate data areas to be maintained without frequent segment register loading (a particularly expensive operation in protect modes).

Direct Color Mode

This section deals with matters unique to the direct color mode of the XGA subsystem.

Palette Loading: It is necessary to load the palette with a fixed set of values. These are described in "Direct Color Mode" on page 3-27.

Coprocessor Support: The XGA coprocessor does not support the 16 bits-per-PEL (bpp) mode. This mode is a display mode only, and must be programmed using the system processor to access the video memory display buffer directly using one of the system video memory apertures (see "System Video Memory Apertures" on page 3-187 and "Memory Access Modes" on page 3-219).

The coprocessor is not disabled in this mode. However, the PEL map formats available for coprocessor operations are restricted to 1, 2, 4, or 8 bpp. The coprocessor can be used in this mode if the application manages the differences in bits per PEL. Some ingenuity is required to achieve useful results using the coprocessor in this way.

Bit Block Transfer Operations
By using the PxBlt operations on an 8-bpp bit map, doubling the dimension width of the bit maps involved, and avoiding arithmetic mixes, bit block transfer operations are possible. Use of the 1-bpp pattern and mask maps are possible if carefully considered and calculated.

Text Operations
Text operations using the coprocessor PxBlt function rely on 1-bpp patterns. By doubling the width of the individual character bit map patterns (interspersing the active bits with zero bits) and writing the high and low order bytes of the required color index separately, text operations are possible.

Section 4. Display Connector

Display Connector Introduction

The synchronization and monitor ID signals are TTL levels. The video signals are analog signals ranging from 0 to 0.7 volts.

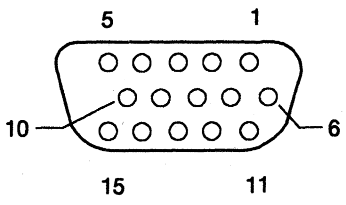


Figure 4-1. Display Connector

		Display Pins	
Pin	**Signal Description**	**Monochrome**	**Color**
1	Red	N/C	Red
2	Green	Mono	Green
3	Blue	N/C	Blue
4	Monitor ID 2		
5	Ground	Self Test	Self Test
6	Red Ground	N/C	Red Ground
7	Green Ground	Mono Ground	Green Ground
8	Blue Ground	N/C	Blue Ground
9	Plug	No Pin	No Pin
10	Ground	Ground	Ground
11	Monitor ID 0		
12	Monitor ID 1		
13	Horizontal Synchronization	Hsync	Hsync
14	Vertical Synchronization	Vsync	Vsync
15	Monitor ID 3		

Figure 4-2. Display Connector Signals

Signal Timing

In the VGA modes, BIOS sets the video subsystem according to the video mode selected. All VGA modes use a 70-Hz vertical retrace except for modes hex 11 and 12. Modes hex 11 and 12 use 60 Hz.

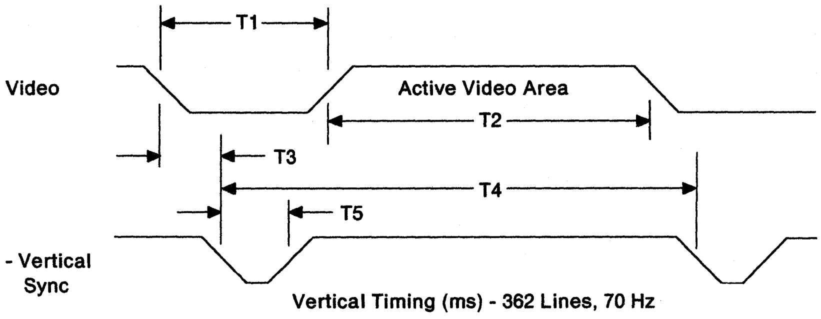
For 1024 x 768 modes, the vertical retrace rate is 43 Hz (this mode is not set by BIOS).

The video subsystem generates the signal timings required by the displays according to the mode selected. The following timing diagrams represent only the vertical frequencies.

Note: The vertical size of the display is encoded using the polarity of the synchronization signals, as shown in the following figure.

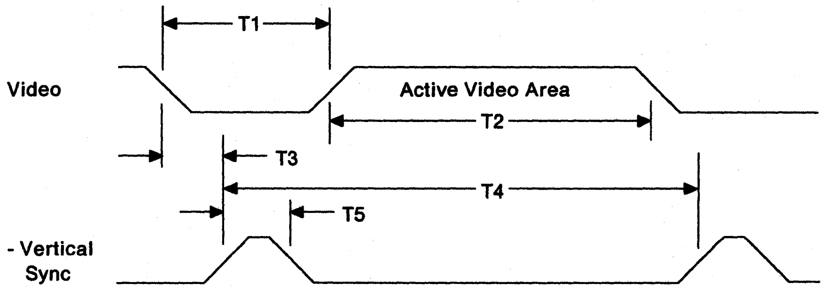
VSYNC Polarity	HSYNC Polarity	Vertical Size
+	+	768 lines (1024 x 768 displays only)
+	–	400 lines
–	+	350 lines
–	–	480 lines

Figure 4-3. Vertical Size of Display



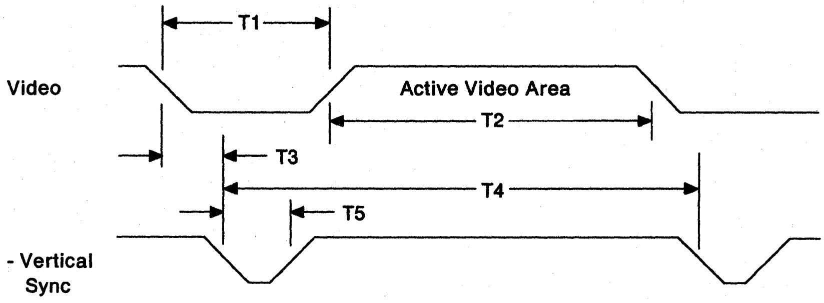
Symbol	Signal Time
T1	2.765 ms
T2	11.504 ms
T3	0.985 ms
T4	14.268 ms
T5	0.064 ms

Figure 4-4. Vertical Timing, 350 Lines



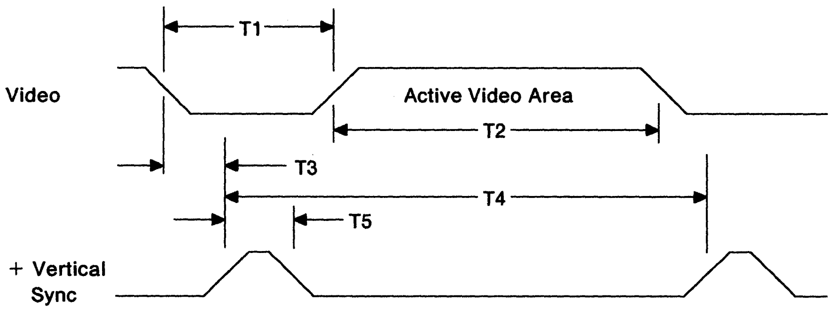
Symbol	Signal Time
T1	1.112 ms
T2	13.156 ms
T3	0.159 ms
T4	14.268 ms
T5	0.064 ms

Figure 4-5. Vertical Timing, 400 Lines



Symbol	Signal Time
T1	0.922 ms
T2	15.762 ms
T3	0.064 ms
T4	16.683 ms
T5	0.064 ms

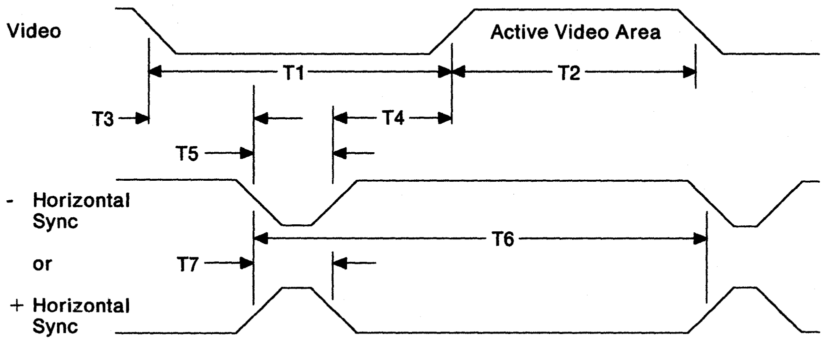
Figure 4-6. Vertical Timing, 480 Lines



Symbol	Signal Times	
T1	0.676 / 0.704 ms	(Odd/Even Lines)
T2	21.620 ms	
T3	0.010 / 0.140 ms	(Odd/Even Lines)
T4	23.000 ms	
T5	0.113 ms	

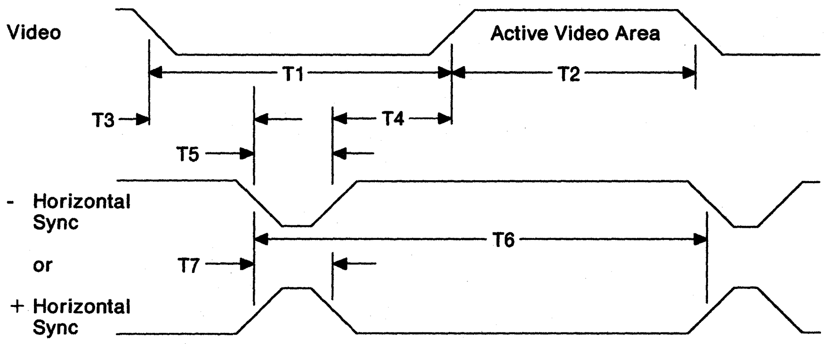
Note: Because this is an interlaced mode, the timing for the odd and even lines is different.

Figure 4-7. Vertical Timing, 768 Lines. Type 1 video does not support these timings.



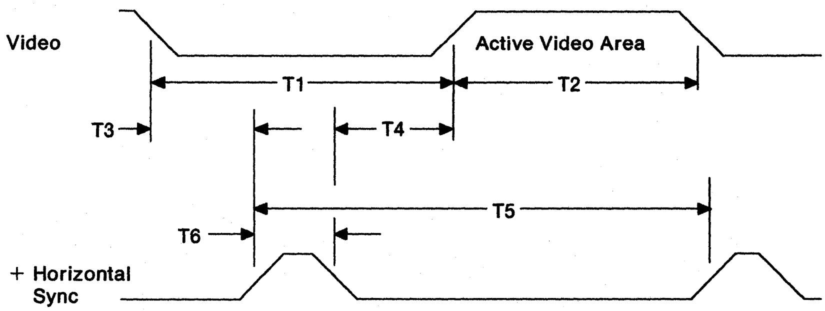
Symbol	Signal Time
T1	5.720 μs
T2	26.058 μs
T3	0.318 μs
T4	1.589 μs
T5	3.813 μs
T6	31.778 μs
T7	3.813 μs

Figure 4-8. Horizontal Timing, 80 Column with Border



Symbol	Signal Time
T1	6.356 μs
T2	25.422 μs
T3	0.636 μs
T4	1.907 μs
T5	3.813 μs
T6	31.778 μs
T7	3.813 μs

Figure 4-9. Horizontal Timing, 40/80 Column, without Border



Symbol	Signal Times
T1	5.35 μs
T2	22.80 μs
T3	0.18 μs
T4	1.25 μs
T5	28.15 μs
T6	3.92 μs

Figure 4-10. Horizontal Timing, 1024 PELs. Type 1 video does not support these timings.

Appendix A. XGA Sample Code

XGA Sample Code Introduction

This appendix contains sample code for the XGA function. It includes code to move from VGA mode to XGA mode. It also includes code to enter 132-Column Text mode while in the XGA function.

Setting the XGA Subsystem into Extended Graphics Mode

Pseudo Code

Main Program

```
• Locate first XGA subsystem with attached display
• If XGA is current system VGA subsystem
    – Chain INT 10h Video handler
    – Chain INT 21h DOS Function handler
    – Chain INT 23h Ctrl Break Exit Address
    – Chain INT 24h Critical Error handler
• If LIM Expanded Memory Manager installed
    – Call LIM Fn 25.Get Physical Address Array
    – Examine returned list for Memory Contention
    – If contention found
        - Display Warning Message
        - Terminate Application
• Chain INT 2Fh Screen Switch Notification handler
• Put XGA in highest Extended Graphics Mode for attached display (see
  "Extended Graphics Mode" on page 3-182)
• Draw simple rectangle ( or whatever )
• Exit
```

Figure A-1 (Part 1 of 3). Extended Graphics Mode Setting PseudoCode

```
INT 10 Handler
    • Examine value of (Ah)
      00h Set Mode
          – Put XGA subsystem in VGA mode (see "VGA Mode" on
            page 3-184)
          – Chain on to saved INT 10h Video Interrupt handler
      0Fh Return current video state
          – Set (AL)=7Fh
          – Interrupt return (IRET)
      Any other value
          – Interrupt return (IRET)
INT 21h DOS Function handler
    • Examine value of (Ah)
      4Ch Program Terminate
          – Put XGA subsystem in VGA mode
          – UnChain and restore original INT 10h Video handler
          – UnChain and restore original INT 21h DOS function
            handler
    • Chain on to saved INT 21h handler
INT 23h Ctrl Break Exit Address
    • Chain on to saved INT 23h handler, using a method that ensures return
      of control through this function handler.
    • On return from chained handler, Examine Carry Flag (CF). If set
      – Put XGA subsystem in VGA mode
      – UnChain and restore original INT 10h Video handler
      – UnChain and restore original INT 21h DOS function handler
    • Interrupt return (IRET)
```

Figure A-1 (Part 2 of 3). Extended Graphics Mode Setting PseudoCode

```
INT 24h Critical Error handler

        • Save video state (or as much as is corrupted by a temporary switch into
          VGA text mode).
        • Put XGA subsystem in VGA mode
        • Chain on to saved INT 24h handler, using a method that ensures return
          of control through this function handler.
        • On return from chained handler, examine AL, as follows:
          0, 1, 3
                  – Put XGA subsytem in Extended graphics mode
                  – Restore video state
          2
                  – UnChain and restore original INT 10h Video handler
                  – UnChain and restore original INT 21h DOS function
                    handler
        • Interrupt return (IRET)

INT 2Fh Screen Switch Notification Handler

        • Examine value of (Ah)
          40h Screen Switch Notification
                  – Examine value of (AL)
                    01h Impending switch to background.
                            - Put XGA subsystem in VGA mode
                            - Chain on to saved INT 2Fh handler (if any)
                    02h Impending switch to foreground
                            - Put XGA subsystem in Extended Graphics
                              Mode
                            - Semaphore redraw required to application
                            - Chain on to saved INT 2Fh handler (if any)
          Any other value
                  – Chain on to saved INT 2F Interrupt vector (if any)
```

Figure A-1 (Part 3 of 3). Extended Graphics Mode Setting PseudoCode

Code Example

Main C Program

```
/* ********************************************************************** */
/*                                                                        */
/*                                                                        */
/*     Program  s_ext                                                     */
/*                                                                        */
/*     Description  This program is sample code to illustrate  entry to   */
/*      Ext Graphics mode and back to VGA mode upon program               */
/*          termination.                                                  */
/*                                                                        */
/* ********************************************************************** */

#define  POS                  0xc4
#define  POS_GET_BASE_ADDRESS 0X00
#define  FALSE                0x00
#define  TRUE                 0x01
#define  EMM_INT              0x67
#define  DEVICE_NAME_LENGTH   0x08
#define  MAX_SLOTS            0x09
#define  XGA_ID_UPPER         0x8fdb
#define  XGA_ID_LOWER         0x8fd8
#define  INDEX_SELECT         0x0a
#define  INDEX_DATA           0x0b
#define  MONITOR_ID           0x52


#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <signal.h>
#include <malloc.h>
#include <memory.h>
#include <stdlib.h>
#include <conio.h>

typedef struct
{
   int      page_segment ;
   int      page_number ;
} MPAA ;                          /* Mapable Physical Address Array struct  */

typedef struct                    /* POS Record */

{
   unsigned int pos_id ;
   char pos_byte1 ;
   char pos_byte2 ;
   char pos_byte3 ;
   char pos_byte4 ;
} POS_REC ;
```

```
union REGS inregs , outregs ;
struct SREGS segregs ;
unsigned int pos_base_address , IoRegBase , slot_number ;

POS_REC  pos_record ;

long int ROS_add_rec[16] = { 0xc0000 ,
                             0xc2000 ,
                             0xc4000 ,
                             0xc6000 ,
                             0xc8000 ,
                             0xca000 ,
                             0xcc000 ,
                             0xce000 ,
                             0xd0000 ,
                             0xd2000 ,
                             0xd4000 ,
                             0xd6000 ,
                             0xd8000 ,
                             0xda000 ,
                             0xdc000 ,
                             0xde000 } ;


unsigned char    vga_data[]      =         { 0x01 , 0x00 , 0x00 ,
                                             0x04 , 0x00 , 0x00 ,
                                             0x05 , 0xff , 0x00 ,
                                             0x0a , 0xff , 0x64 ,
                                             0x0a , 0x15 , 0x50 ,
                                             0x0a , 0x14 , 0x50 ,
                                             0x0a , 0x00 , 0x51 ,
                                             0x0a , 0x04 , 0x54 ,
                                             0x0a , 0x7f , 0x70 ,
                                             0x0a , 0x20 , 0x2a ,
                                             0x00 , 0x01 , 0x00
                                           } ;

unsigned char    nm_data[]       =         { 0x04 , 0x00 , 0x00 , 0x00 ,
                                             0x05 , 0xff , 0xff , 0x00 ,
                                             0x00 , 0x04 , 0x04 , 0x00 ,
                                             0x0a , 0x00 , 0x00 , 0x64 ,
                                             0x01 , 0x00 , 0x00 , 0x00 ,
                                             0x08 , 0x00 , 0x00 , 0x00 ,
                                             0x06 , 0x00 , 0x00 , 0x00 ,
                                             0x09 , 0x03 , 0x02 , 0x00 ,
                                             0x0a , 0x01 , 0x01 , 0x50 ,
                                             0x0a , 0x00 , 0x00 , 0x50 ,
                                             0x0a , 0x9d , 0x9d , 0x10 ,
                                             0x0a , 0x00 , 0x00 , 0x11 ,
                                             0x0a , 0x7f , 0x7f , 0x12 ,
                                             0x0a , 0x00 , 0x00 , 0x13 ,
                                             0x0a , 0x7f , 0x7f , 0x14 ,
                                             0x0a , 0x00 , 0x00 , 0x15 ,
                                             0x0a , 0x9d , 0x9d , 0x16 ,
                                             0x0a , 0x00 , 0x00 , 0x17 ,
                                             0x0a , 0x87 , 0x87 , 0x18 ,
                                             0x0a , 0x00 , 0x00 , 0x19 ,
                                             0x0a , 0x9c , 0x9c , 0x1a ,
                                          0x0a , 0x00 , 0x00 , 0x1b ,
                                          0x0a , 0x40 , 0x40 , 0x1c ,
                                          0x0a , 0x04 , 0x04 , 0x1e ,
                                          0x0a , 0x30 , 0x30 , 0x20 ,
                                          0x0a , 0x03 , 0x03 , 0x21 ,
                                          0x0a , 0xff , 0xff , 0x22 ,
                                          0x0a , 0x02 , 0x02 , 0x23 ,
                                          0x0a , 0xff , 0xff , 0x24 ,
                                          0x0a , 0x02 , 0x02 , 0x25 ,
                                          0x0a , 0x30 , 0x30 , 0x26 ,
                                          0x0a , 0x03 , 0x03 , 0x27 ,
                                          0x0a , 0x00 , 0x00 , 0x28 ,
                                          0x0a , 0x03 , 0x03 , 0x29 ,
                                          0x0a , 0x08 , 0x08 , 0x2a ,
                                          0x0a , 0xff , 0xff , 0x2c ,
                                          0x0a , 0xff , 0xff , 0x2d ,
                                          0x0a , 0x00 , 0x00 , 0x36 ,
                                          0x0a , 0x00 , 0x00 , 0x40 ,
                                          0x0a , 0x00 , 0x00 , 0x41 ,
                                          0x0a , 0x00 , 0x00 , 0x42 ,
                                          0x0a , 0x80 , 0x40 , 0x43 ,
                                          0x0a , 0x00 , 0x00 , 0x44 ,
                                          0x0a , 0x0d , 0x0d , 0x54 ,
                                          0x0a , 0x03 , 0x02 , 0x51 ,
                                          0x0a , 0x00 , 0x00 , 0x70 ,
                                          0x0a , 0x0f , 0x0f , 0x50 ,
                                          0x0a , 0x00 , 0x00 , 0x55 ,
                                          0x0a , 0x00 , 0x00 , 0x60 ,
                                          0x0a , 0x00 , 0x00 , 0x61 ,
                                          0x0a , 0x00 , 0x00 , 0x62 ,
                                          0x0a , 0x00 , 0x00 , 0x63 ,
                                          0x0a , 0xff , 0xff , 0x64
                                          } ;

 /* colour_default_palette                  R      G      B   */

unsigned char  colour_default_palette[] = { 0x00 , 0x00 , 0x00 ,
                                          0x00 , 0x00 , 0xa8 ,
                                          0x00 , 0xa8 , 0x00 ,
                                          0x00 , 0xA8 , 0xA8 ,
                                          0xA8 , 0x00 , 0x00 ,
                                          0xA8 , 0x00 , 0xa8 ,
                                          0xA8 , 0x54 , 0x00 ,
                                          0xA8 , 0xA8 , 0xA8 ,
                                          0x54 , 0x54 , 0x54 ,
                                          0x54 , 0x54 , 0xfc ,
                                          0x54 , 0xfc , 0x54 ,
                                          0x54 , 0xFC , 0xFC ,
                                          0xFC , 0x54 , 0x54 ,
                                          0xFC , 0x54 , 0xFC ,
                                          0xFC , 0xFC , 0x54 ,
                                          0xFC , 0xFC , 0xFC } ;


int      IpInt , cop_instance ;
long int ROS_address  ;
char     XGAFound , VRAM_1Meg;
char     XGAInVGA , ExtG_mode_set , slot_enabled ;
void far *int_10_original_vector ;
void far *int_2f_original_vector ;
FILE   *stream ;
int      delaytime = 1000 ;
char VramIr , A1024x768 ;

unsigned short int VRAM_address_lo , VRAM_address_hi ;


interrupt far int_10( void ) ;
interrupt far int_2f( void ) ;
char *build_ptr( unsigned int , unsigned int );
char emm_installed( void ) ;
int far return_ax( void ) ;
int far return_7f( void ) ;
void PutXGAInVGA   (void) ;
void PutXGAInExtG(void);
int exit_handler( void ) ;
void signal_handler( void ) ;
void co_pro_blit(void);
void CoProWriteByte(int , unsigned char );
void CoProWriteWord(int , unsigned int );
void CoProPrintByte( int , char * );
void WaitForCoProReady(void);
void delay(long int);

/* ************************************************************************** */
/*                                                                            */
/*                                                                            */
/*     Function main()                                                        */
/*                                                                            */
/*     Description  This is the program entry point.                          */
/*                                                                            */
/*                                                                            */
/* ************************************************************************** */

void main( void )
{
   int      index ;
   MPAA     *mpaa ;
   unsigned int  ipdata ;
   unsigned char far *vram_address ;
   unsigned char ip_byte ;

   atexit( exit_handler );
   signal( SIGINT  , signal_handler );
   signal( SIGFPE  , signal_handler );
   signal( SIGABRT , signal_handler );


   ExtG_mode_set = FALSE ;
   int_10_original_vector = 0 ;
   int_2f_original_vector = 0 ;

   inregs.h.ah = POS ;                             /* get base POS address */
   inregs.h.al = POS_GET_BASE_ADDRESS ;            /* from system services */
   int86( 0x15 , &inregs , &outregs );
```

```
pos_base_address = outregs.x.dx ;
XGAFound = FALSE ;
if( outregs.x.cflag )                          /* carry flag set means */
{                                              /* not a microchannel */
   printf( "No XGA Installed\n" ) ;            /* machine          */
}
else
{
   XGAFound = FALSE ;
   slot_enabled = FALSE ;
   for ( slot_number = 0 ; slot_number  <= MAX_SLOTS ; slot_number++ )
   {
      _disable();                      /* Disable interrupts */
      if (slot_number == 0)
      { /* Look at the planar for XGA */
         outp( 0X094 , 0x0df ) ;       /*  Enable planar for setup  */
      }
      else
      { /* Look in the slots for XGA */
         inregs.x.ax   = 0xc401 ;
         inregs.x.bx   = slot_number ;
         int86( 0x15 , &inregs , &outregs ) ;  /* enable slot for update */
      }
      slot_enabled = TRUE ;
      /* Get pos record for the slot */
      pos_record.pos_id = inpw( pos_base_address ) ;
      pos_record.pos_byte1 = (char)inp( pos_base_address + 2 ) ;
      pos_record.pos_byte2 = (char)inp( pos_base_address + 3 ) ;
      pos_record.pos_byte3 = (char)inp( pos_base_address + 4 ) ;
      pos_record.pos_byte4 = (char)inp( pos_base_address + 5 ) ;
      IoRegBase = (( pos_record.pos_byte1 & 0x0e ) << 3 ) + 0x2100 ;

      if(slot_number == 0)
      {
         outp( 0X094 , 0x0ff ) ;       /*  Enable planar for normal mode */
      }
      else
      {
         inregs.x.ax   = 0xc402 ;
         inregs.x.bx   = slot_number ;
         int86( 0x15 , &inregs , &outregs ) ;  /* enable slot normal mode */
      }
      slot_enabled = FALSE ;
      _enable();                       /* Enable interrupts */
      /* Check for a valid XGA POS id */
      if ( pos_record.pos_id >= XGA_ID_LOWER &&
           pos_record.pos_id <= XGA_ID_UPPER )
      {
         /* XGA found in slot */

         /* Look to see if display connected to XGA */
         outp( IoRegBase + INDEX_SELECT , MONITOR_ID );
         if ( ( inp(IoRegBase + INDEX_DATA) & 0x0f) != 0x0F )
         {
            /* Display connected to XGA */
            XGAFound = TRUE ;
```

```
/* Determine if XGA in VGA */
ipdata = inp( IoRegBase ) ;
if( ipdata & 0x01 )
{
   XGAInVGA = TRUE ;
   /* Chain Int 10H */
   int_10_original_vector = _dos_getvect( 0x10 ) ;
   _dos_setvect( 0x10 , int_10 ) ;
}
else
   XGAInVGA = FALSE ;
/* calculate VRAM address */
VRAM_address_lo = 0x0 ;
VRAM_address_hi =  ((short int)( pos_record.pos_byte3 & 0xfe )) <<
VRAM_address_hi |= ((short int)(pos_record.pos_byte1 & 0x0e)) << 5

/* check VRAM  */
outp(IoRegBase + 4 , 0x00 );
outp(IoRegBase + 0 , 0x04 );
outp(IoRegBase + 1 , 0x01 );
outpw(IoRegBase + 0x0a , 0x0064 );


VRAM_1Meg = FALSE ;
outp(IoRegBase + 8 , 0x0c) ;
vram_address = (unsigned char far *)0xa0000000 ;
*vram_address = 0xa5 ;

vram_address++ ;
*vram_address = 0 ;

vram_address-- ;
ip_byte = *vram_address ;

if(ip_byte == 0xa5)
{
   VRAM_1Meg = TRUE ;
}

index  = (pos_record.pos_byte1 >> 4) & 0xf ;
cop_instance = (pos_record.pos_byte1 >> 1) & 0x07 ;
ROS_address = ROS_add_rec[ index ] ;
if( emm_installed() )
{
   /* get number of entries in mappable physical page */
   inregs.x.ax = 0x5801 ;
   int86( 0x15 , &inregs , &outregs ) ;
   if(outregs.x.cx)
   {
      /* entries exist in the list */
      mpaa = (MPAA *)calloc( outregs.x.cx , sizeof( MPAA ) ) ;
      if(!mpaa)
      {
         printf("Unable to allocate memory\n");
      }
      else
      {
         segregs.es  = FP_SEG( mpaa ) ;
```

```
                        inregs.x.di = FP_OFF( mpaa ) ;
                        int86x( 0x67 , &inregs , &outregs , &segregs ) ;

                        for( ; outregs.x.cx > 0 ; mpaa++ )
                        {
                           if(ROS_address ==  mpaa->page_segment )
                           {
                              printf("Extended memory conflict at segement");
                              printf(" address %x\n" , mpaa->page_segment );
                              exit(0);
                           }
                        }
                     }
                  }
               }
               /* Chain Int 2fH */
               int_2f_original_vector = _dos_getvect( 0x2f ) ;
               _dos_setvect( 0x2f , int_2f ) ;

               PutXGAInExtG();

               co_pro_blit();
            }
         }
         if( XGAFound ) break ;
      }
   }
}

/* ************************************************************************ */
/*                                                                          */
/*       Function    -   pointer = build_ptr( segment , offset )            */
/*                                                                          */
/*       Description -   Constructs a pointer from segment and offset.      */
/*                                                                          */
/*                                                                          */
/*                                                                          */
/* ************************************************************************ */
char *build_ptr( unsigned int segment , unsigned int offset )
{
    char *ptr ;

    ptr = (char *)(((unsigned long)segment << 16) + offset ) ;
    return( ptr ) ;
}

/* ************************************************************************ */
/*                                                                          */
/*       Function    -   status = emm_installed()                           */
/*                                                                          */
/*       Description -   checks to see if extended memory has been installed.  */
/*                                                                          */
/*                                                                          */
/* ************************************************************************ */
char emm_installed( )
{
    char *EMM_device_name = "EMMXXXX0" ;
    char *int_67_device_name_ptr ;
    inregs.h.ah = 0x35 ;
    inregs.h.al = EMM_INT ;
    intdosx( &inregs , &outregs , &segregs ) ;
    int_67_device_name_ptr = build_ptr( segregs.es , 0x0a ) ;
    if( memcmp(EMM_device_name , int_67_device_name_ptr , DEVICE_NAME_LENGTH ))
      return ( FALSE );
    else
      return ( TRUE ) ;
}


/* ************************************************************************* */
/*                                                                           */
/*      Function     -   interrupt routine int_10()                          */
/*                                                                           */
/*      Description -   internal INT 10h interrupt handler                   */
/*                                                                           */
/*                                                                           */
/* ************************************************************************* */
interrupt far int_10( void )
{
   unsigned int ax ;

   ax = return_ax() >> 8 ;
   if( ax == 0 )
   {
      /* set xga to vga mode */
      _chain_intr( int_10_original_vector ) ;
   }
   else if ( ax == 0x0f )
   {
      return_7f() ;
   }
   else
   {
      _chain_intr( int_10_original_vector ) ;
   }
}
```

```
/* ************************************************************************ */
/*                                                                          */
/*     Function    -   Interrupt routine int_2f()                           */
/*                                                                          */
/*     Description -   Internal INT 2fh interrupt handler                   */
/*                                                                          */
/*                                                                          */
/* ************************************************************************ */
interrupt far int_2f( void )
{
   unsigned int     ax ;
   unsigned char    ah ;
   unsigned char    al ;

   ax = return_ax() ;
   ah = (unsigned char)(ax >> 8) ;
   al = (unsigned char)(ax & 0x0f);
   if( ah == 0x40 )
   {  /* Screen switch notification received */
      if (al == 0x01 )     /* Switch to background */
         PutXGAInVGA();
      else if (al == 0x02) /* Switch to foreground */
         PutXGAInExtG();
   }
   if(int_2f_original_vector)   _chain_intr( int_2f_original_vector ) ;
}

/* ************************************************************************ */
/*                                                                          */
/*     Function    -   signal_handler()                                     */
/*                                                                          */
/*     Description -   error handler                                        */
/*                                                                          */
/*                                                                          */
/* ************************************************************************ */
void signal_handler( void )
{
   exit(0);
}

/* ************************************************************************ */
/*                                                                          */
/*     Function    -   exit_handler()                                       */
/*                                                                          */
/*     Description -   Exit handler                                         */
/*                                                                          */
/* ************************************************************************ */
int exit_handler( void )
{
   /* reset interrupt vectors to original values */
   if ( int_10_original_vector ) _dos_setvect( 0x10 , int_10_original_vector ) ;
   if ( int_2f_original_vector ) _dos_setvect( 0x2f , int_2f_original_vector ) ;

   /* reset to VGA if originally in VGA mode */
   if ( ExtG_mode_set && XGAInVGA ) PutXGAInVGA     () ;
   printf ("Completed\n");
   return (0) ;
}
```

```
/* ************************************************************************ */
/*                                                                          */
/*      Function    -   co_pr_blit()                                        */
/*                                                                          */
/*      Description -   Use coprocessor to blit to the display              */
/*                                                                          */
/* ************************************************************************ */
void co_pro_blit(void)
{
    unsigned char i , j , k;

    A1024x768 = TRUE ;

    CoProWriteByte( 0x11 , 0x00 ) ;
    CoProWriteByte( 0x12 , 0x01  ) ;

    /* set up VRAM base address */
    CoProWriteWord( 0x14 , VRAM_address_lo );
    CoProWriteWord( 0x16 , VRAM_address_hi );

    CoProWriteWord( 0x18 , A1024x768 ? 0x03ff : 0x02a8 ) ;
    CoProWriteWord( 0x1a , A1024x768 ? 0x02ff : 0x01e0 ) ;
    CoProWriteByte( 0x1c , (unsigned char)(VRAM_1Meg ? 0x03 : 0x02 )) ;

    CoProWriteByte( 0x48 , 0x03 )  ;
    CoProWriteByte( 0x49 , 0x05 )  ;
    CoProWriteByte( 0x4a , 0x04 )  ;
    CoProWriteWord( 0x50 , 0xff ) ;
    CoProWriteWord( 0x54 , 0x7f ) ;
    CoProWriteByte( 0x58 , 0x00 )  ;
    CoProWriteByte( 0x5c , 0x00 )  ;
    CoProWriteWord( 0x60 , A1024x768 ? 0x03ff : 0x02a8 ) ;
    CoProWriteWord( 0x62 , A1024x768 ? 0x02ff : 0x01e0 ) ;
    CoProWriteWord( 0x6c , 0x0000 ) ;
    CoProWriteWord( 0x6e , 0x0000 ) ;
    CoProWriteWord( 0x70 , 0x0000 ) ;
    CoProWriteWord( 0x72 , 0x0000 ) ;
    CoProWriteWord( 0x74 , 0x0000 ) ;
    CoProWriteWord( 0x76 , 0x0000 ) ;
    CoProWriteWord( 0x78 , 0x0000 ) ;
    CoProWriteWord( 0x7a , 0x0000 ) ;
    CoProWriteWord( 0x7c , 0x8000 ) ;
    CoProWriteWord( 0x7e , 0x0811 ) ;

    WaitForCoProReady() ;

    CoProWriteWord( 0x60 , 0x0015 ) ;
    CoProWriteWord( 0x62 , 0x02ff ) ;
    CoProWriteWord( 0x60 , A1024x768 ? 0x0015 : 0x0010 ) ;
    CoProWriteWord( 0x62 , A1024x768 ? 0x02ff : 0x02ff ) ;
    for (j=0; j < 30 ; j++)
    {
       k=0;
       for (i=1; i<= 34  ; i++)
       {
           if (k > 15 )
              k = 1 ;
           else
```

```
                k++ ;
             CoProWriteWord( 0x78 , (0x0030 * i) - 15 +j) ;
             CoProWriteWord( 0x7a , 0x0000 ) ;
             CoProWriteByte( 0x58 , k ) ;
             CoProWriteByte( 0x7f ,0x08 ) ;
             WaitForCoProReady() ;
          }
      }
      CoProWriteWord( 0x60 , 0x03ff ) ;
      CoProWriteWord( 0x62 , 0x02ff ) ;
      CoProWriteWord( 0x70 , 0x0015 ) ;
      CoProWriteWord( 0x72 , 0x0000 ) ;
      CoProWriteWord( 0x78 , 0x0030 ) ;
      CoProWriteWord( 0x7a , 0x0024 ) ;
      CoProWriteByte( 0x7f ,0x28 ) ;
      WaitForCoProReady() ;

}

/* ************************************************************************** */
/*                                                                            */
/*     Function    -  CoProWriteByte(offset , data)                           */
/*                                                                            */
/*     Description -  Writes a data byte to the coprocessor at the            */
/*                    supplied offset.                                        */
/*                                                                            */
/* ************************************************************************** */
void CoProWriteByte(int offset , unsigned char ipdata)
{
   unsigned char far *cop ;
   unsigned long      tmp ;

   tmp = ROS_address + 0x1c00 + offset + (cop_instance * 128 ) ;
   cop = (void far *)(((tmp & 0xffff0 )<<12 ) + (tmp & 0x0f)) ;
   *cop= ipdata ;
}
void CoProPrintByte(int offset , char *string)
{
   unsigned char far *cop ;
   unsigned long      tmp ;

   tmp = ROS_address + 0x1c00 + offset + (cop_instance * 128 ) ;
   cop = (void far *)(((tmp & 0xffff0 )<<12 ) + (tmp & 0x0f)) ;
   printf(string);
   printf("   -   %x\n",*cop);
}
```

```
/* ************************************************************************ */
/*                                                                          */
/*      Function    -   CoProWriteWord(offset , data)                       */
/*                                                                          */
/*      Description -   Writes a data word to the coprocessor at the        */
/*                      supplied offset.                                    */
/*                                                                          */
/* ************************************************************************ */

void CoProWriteWord(int offset , unsigned int ipdata)
{
    unsigned int far *cop ;
    unsigned long    tmp ;

    tmp = ROS_address + 0x1c00 + offset + (cop_instance * 128 ) ;
    cop = (void far *)(((tmp & 0xffff0 )<<12 ) + (tmp & 0x0f)) ;
    *cop= ipdata ;
}


/* ************************************************************************ */
/*                                                                          */
/*      Function    -   WaitForCoProReady()                                 */
/*                                                                          */
/*      Description -   Waits until coprocessor in  ready state             */
/*                                                                          */
/* ************************************************************************ */
void WaitForCoProReady(void)
{
    unsigned char far       *cop ;
    unsigned long           tmp ;
    long int                count ;

    tmp = ROS_address + 0x1c00 + 0x11 + (cop_instance * 128 ) ;
    cop = (void far *)(((tmp & 0xffff0 )<<12 ) + (tmp & 0x0f)) ;
    count = 0 ;
    for(;;)
    {
        if ( ((*cop & 0x80)==0) || (count > 20000)) break;
        count++;
    }
}
```

```
/* ************************************************************************ */
/*                                                                          */
/*      Function    -   PutXGAInExtG()                                      */
/*                                                                          */
/*      Description -   Puts the XGA into Extended Graphics Mode            */
/*                                                                          */
/* ************************************************************************ */
void PutXGAInExtG( void )
{
   int    i , res , palette_size ;

   outp( 0x03c3 , 0x01 );
   ExtG_mode_set = TRUE ;
   /* 1MB VRAM res = 1 if 512KB VRAM res = 2 */
   res = VRAM_1Meg ? 1 : 2 ;
   for (i = 0 ; i < sizeof(nm_data) ; i = i + 4 )
   {
      if (nm_data[i+3])
         outpw( IoRegBase + nm_data[i] ,
                (((int)nm_data[i+res]) << 8) + (unsigned)nm_data[i+3] ) ;
      else
         outp( IoRegBase + nm_data[i] , (int)nm_data[i+res] );
   }

   outpw( IoRegBase + 0x0a , 0x0066 );
   outpw( IoRegBase + 0x0a , 0x0060 );
   outpw( IoRegBase + 0x0a , 0x0061 );

   palette_size = sizeof(colour_default_palette) ;
   for ( i=0 ; i <= palette_size ; i++ )
   {
      /* select palette data register */
      outp( IoRegBase + INDEX_SELECT , 0x65 ) ;
      outp( IoRegBase + 0x0b , (int)colour_default_palette[i] ) ;
   }

}
```

```
/* *********************************************************************** */
/*                                                                         */
/*     Function     -   PutXGAInVGA()                                      */
/*                                                                         */
/*     Description -   puts  the XGA into VGA mode                         */
/*                                                                         */
/* *********************************************************************** */
void PutXGAInVGA( void )
{
   int    i ;

   vram_address = (unsigned char far *)0xA0000000;

   /* clear 1st 256KB of VRAM */
   for (i = 0 ; i < 4 ; i++)
   {
      outp(IoRegBase + 8 , i);
      ptr = vram_address ;
      memset(ptr , 0 , 0x8000);  /* set 1st 32KB of 64KB aperture*/
      ptr = vram_address + 0x8000 ;
      memset(ptr , 0 , 0x8000);  /* set 2nd 32KB of 64KB aperture*/
   }

   for (i = 0 ; i < sizeof(vga_data) ; i = i + 3 )
   {
      if (vga_data[i+2])
         outpw( IoRegBase + vga_data[i] ,
                (((unsigned)vga_data[i+1]) << 8) + (unsigned)vga_data[i+2] ) ;
      else
         outp( IoRegBase + vga_data[i] , (int)vga_data[i+1] );
   }

   outp( 0x03c3 , 0x01 );


   /* select scan lines for alphanumeric modes */
   inregs.h.ah = 0x12 ;
   inregs.h.al = 0x02 ;     /* 400 scan lines */
   inregs.h.bl = 0x30 ;
   int86( 0x10 , &inregs , &outregs ) ;

   inregs.h.ah = 0x00 ;
   inregs.h.al = 0x03 ;
   int86( 0x10 , &inregs , &outregs ) ;

}
```

```
;***************************************************************************
;**                                                                       **
;**      Function        _return_ax()                                     **
;**                                                                       **
;**      Description     unwinds the stack after a call to an interrupt   **
;**                      routine to obtain contents of ax register, the   **
;**                      value of which is returned. The stack is restored **
;**                      prior to the return.                             **
;**                                                                       **
;***************************************************************************

.286c
.MODEL  SMALL
.DATA
return_segment_address     dw  ?
return_offset_address      dw  ?
ret_data_seg               dw  ?
ret_bp                     dw  ?

.CODE
        PUBLIC  _return_ax
_return_ax      PROC FAR
        mov     ret_bp , bp
        pop     bx
        pop     es
        mov     return_segment_address , es
        mov     return_offset_address , bx
        add     sp,+2
        pop     es
        pop     bx ; data seg
        mov     ret_data_seg , bx
        popa
        pusha
        mov     bx,ret_data_seg
        push    bx ; data seg
        push    es
        sub     sp,+2
        mov     es,return_segment_address
        mov     bx,return_offset_address
        push    es
        push    bx
        mov     bp , ret_bp
        ret
_return_ax      ENDP

        END
```

```
;*****************************************************************************
;**                                                                         **
;**      Function       return_7f                                           **
;**                                                                         **
;**      Description    unwinds the stack after a call to an interrupt      **
;**                     routine to obtain contents of ax register, the      **
;**                     value of which is returned. The stack is restored   **
;**                     prior to the return.                                **
;**                                                                         **
;*****************************************************************************

.286c
.MODEL  SMALL
.DATA
return_segment_address      dw  ?
return_offset_address       dw  ?
ret_data_seg                dw  ?
ret_bp                      dw  ?

.CODE
        PUBLIC  _return_7f
_return_7f      PROC FAR
        mov     ret_bp , bp
        pop     bx
        pop     es
        mov     return_segment_address , es
        mov     return_offset_address , bx
        add     sp,+2
        pop     es
        pop     bx ; data seg
        mov     ret_data_seg , bx
        popa
        mov     ah,7fh
        pusha
        mov     bx,ret_data_seg
        push    bx ; data seg
        push    es
        sub     sp,+2
        mov     es,return_segment_address
        mov     bx,return_offset_address
        push    es
        push    bx
        mov     bp , ret_bp
        ret
_return_7f      ENDP

        END
```

Setting the XGA Subsystem into 132-Column Text Mode

Pseudo Code

```
Main Program

        • Locate XGA subsystem with attached display in VGA mode
        • If none, display error message and return.
        • Chain INT 10h Video handler
        • Chain INT 21h DOS Function handler
        • Chain INT 23h Ctrl Break Exit Address
        • Chain INT 2Fh Screen Switch Notification handler
        • Put XGA into 132-column text mode (see "132-Column Text Mode" on
          page 3-185)
        • Display simple text
        • Exit

INT 10 Handler

        • Examine value of (Ah)
          00h Set Mode
                    – Put XGA subsystem in normal VGA mode (see "VGA
                      Mode" on page 3-184)
                    – Chain on to saved INT 10h Video Interrupt handler.
          Any other value
                    – Interrupt return (IRET)
```

Figure A-2 (Part 1 of 2). 132-column Text Mode Setting Pseudo Code

```
INT 21h DOS Function handler
        • Examine value of (Ah)
          4Ch Program Terminate
                   – Put XGA subsystem in normal VGA mode
                   – UnChain and restore original INT 10h Video handler
                   – UnChain and restore original INT 21h DOS function
                     handler
        • Chain on to saved INT 21h handler
INT 23h Ctrl Break Exit Address
        • Chain on to saved INT 23h handler, using a method that ensures return
          of control through this function handler.
        • On return from chained handler, examine Carry Flag (CF). If set
          – Put XGA subsystem in normal VGA mode
          – UnChain and restore original INT 10h Video handler
          – UnChain and restore original INT 21h DOS function handler
        • Interrupt return (IRET)
INT 2Fh Screen Switch Notification Handler
        • Examine value of (Ah)
          40h Screen Switch Notification
                   – Examine value of (AL)
                     01h Impending switch to background.
                              - Put XGA subsystem in normal VGA mode
                              - Chain on to saved INT 2Fh handler (if any)
                     02h Impending switch to foreground
                              - Put XGA subsystem in 132-column text
                                mode
                              - Semaphore redraw required to application
                              - Chain on to saved INT 2Fh handler (if any)
          Any other value
                   – Chain on to saved INT 2F Interrupt vector (if any)
```

Figure A-2 (Part 2 of 2). 132-column Text Mode Setting Pseudo Code

Code Example

Main C Program

```
/* ************************************************************************* */
/*                                                                           */
/*                                                                           */
/*     Program  s_132n                                                       */
/*                                                                           */
/*     Description  This program is sample code to illustrate  entry to      */
/*          132-column text mode and back to VGA mode upon program           */
/*          termination.                                                     */
/*                                                                           */
/* ************************************************************************* */

#define  POS                   0xc4
#define  POS_GET_BASE_ADDRESS  0X00
#define  FALSE                 0x00
#define  TRUE                  0x01
#define  EMM_INT               0x67
#define  DEVICE_NAME_LENGTH    0x08
#define  MAX_SLOTS             0x09
#define  XGA_ID_UPPER          0x8fdb
#define  XGA_ID_LOWER          0x8fd8
#define  INDEX_SELECT          0x0a
#define  INDEX_DATA            0x0b
#define  MONITOR_ID            0x52


#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <signal.h>
#include <malloc.h>
#include <memory.h>
#include <stdlib.h>
#include <conio.h>

typedef struct                      /* POS Record */
{
   unsigned int pos_id ;
   char pos_byte1 ;
   char pos_byte2 ;
   char pos_byte3 ;
   char pos_byte4 ;
} POS_REC ;
```

```
union REGS inregs , outregs ;
struct SREGS segregs ;
unsigned int pos_base_address , IoRegBase , slot_number ;

POS_REC  pos_record ;

unsigned char    vga_data[]      =          { 0x01 , 0x00 , 0x00 ,
                                              0x04 , 0x00 , 0x00 ,
                                              0x05 , 0xff , 0x00 ,
                                              0x0a , 0xff , 0x64 ,
                                              0x0a , 0x15 , 0x50 ,
                                              0x0a , 0x14 , 0x50 ,
                                              0x0a , 0x00 , 0x51 ,
                                              0x0a , 0x04 , 0x54 ,
                                              0x0a , 0x7f , 0x70 ,
                                              0x0a , 0x20 , 0x2a ,
                                              0x00 , 0x01 , 0x00
                                            } ;

char     RedrawRequired ;
char     XGAFound ;
char     XGAInVGA ;
void far *int_10_original_vector ;
void far *int_2f_original_vector ;

interrupt far int_10( void ) ;
interrupt far int_2f( void ) ;
int far return_ax( void ) ;
int far return_7f( void ) ;
void PutXGAInVGA(void) ;
void PutXGAIn132(void);
int exit_handler( void ) ;
void signal_handler( void ) ;
void DisplayText( void );
```

```
/* ************************************************************************ */
/*                                                                          */
/*                                                                          */
/*    Function main()                                                       */
/*                                                                          */
/*    Description  This is the program entry point.                        */
/*                                                                          */
/*                                                                          */
/* ************************************************************************ */

void main( void )
{
   int      index ;
   unsigned int  ipdata ;
   unsigned char far *vram_address ;
   unsigned char ip_byte ;

   atexit( exit_handler );
   signal( SIGINT  , signal_handler );
   signal( SIGFPE  , signal_handler );
   signal( SIGABRT , signal_handler );


   int_10_original_vector = 0 ;
   int_2f_original_vector = 0 ;

   inregs.h.ah = POS ;                              /* get base POS address */
   inregs.h.al = POS_GET_BASE_ADDRESS ;             /* from system services */
   int86( 0x15 , &inregs , &outregs );

   pos_base_address = outregs.x.dx ;
   XGAFound = FALSE ;
   if( outregs.x.cflag )                            /* carry flag set means */
   {                                                /* not a microchannel */
      printf( "No XGA Installed\n" ) ;              /* machine          */
   }
   else
   {
      XGAFound = FALSE ;
      XGAInVGA = FALSE ;

      for ( slot_number = 0 ; slot_number  <= MAX_SLOTS ; slot_number++ )
      {
         _disable();                     /* Disable interrupts */
         if (slot_number == 0)
         { /* Look at the planar for XGA */
            outp( 0X094 , 0x0df ) ;      /*  Enable planar for setup  */
         }
         else
         { /* Look in the slots for XGA */
            inregs.x.ax    = 0xc401 ;
            inregs.x.bx    = slot_number ;
            int86( 0x15 , &inregs , &outregs ) ;  /* enable slot for update */
         }

         /* Get pos record for the slot */
         pos_record.pos_id = inpw( pos_base_address ) ;
      pos_record.pos_byte1 = (char)inp( pos_base_address + 2 ) ;
      pos_record.pos_byte2 = (char)inp( pos_base_address + 3 ) ;
      pos_record.pos_byte3 = (char)inp( pos_base_address + 4 ) ;
      pos_record.pos_byte4 = (char)inp( pos_base_address + 5 ) ;
      IoRegBase = (( pos_record.pos_byte1 & 0x0e ) << 3 ) + 0x2100 ;

      if(slot_number == 0)
      {
         outp( 0X094 , 0x0ff ) ;     /*  Enable planar for normal mode */
      }
      else
      {
         inregs.x.ax   = 0xc402 ;
         inregs.x.bx   = slot_number ;
         int86( 0x15 , &inregs , &outregs ) ;  /* enable slot normal mode */
      }

      _enable();                      /* Enable interrupts */
      /* Check for a valid XGA POS id */
      if ( pos_record.pos_id >= XGA_ID_LOWER &&
           pos_record.pos_id <= XGA_ID_UPPER )
      {
         /* XGA found in slot */

         /* Look to see if display connected to XGA */
         outp( IoRegBase + INDEX_SELECT , MONITOR_ID );
         if ( ( inp(IoRegBase + INDEX_DATA) & 0x0f) != 0x0F )
         {
            /* Display connected to XGA */
            XGAFound = TRUE ;

            /* Determine if XGA in VGA */
            ipdata = inp( IoRegBase ) ;
            if( ipdata & 0x01 )
            {
               XGAInVGA = TRUE ;
               break;
            }
         }
      }
   }
   if (XGAInVGA == FALSE )
   {
       printf( "XGA in VGA with attached display - not found\n");
   }
   else
   {

      RedrawRequired = FALSE ;

      /* Chain Int 10H */
      int_10_original_vector = _dos_getvect( 0x10 ) ;
      _dos_setvect( 0x10 , int_10 ) ;

      /* Chain Int 2fH */
      int_2f_original_vector = _dos_getvect( 0x2f ) ;
      _dos_setvect( 0x2f , int_2f ) ;
```

```
            PutXGAIn132();

            DisplayText();
         }
      }
   }

/* ********************************************************************** */
/*                                                                        */
/*       Function     -   interrupt routine int_10()                      */
/*                                                                        */
/*       Description -    internal INT 10h interrupt handler              */
/*                                                                        */
/*                                                                        */
/* ********************************************************************** */
interrupt far int_10( void )
{
   int ax ;

   ax = return_ax() >> 8 ;
   if( ax == 0 )
   {
      /* set xga to vga mode */
      PutXGAInVGA();
      _chain_intr( int_10_original_vector ) ;
   }
   else if ( ax == 0x0f )
   {
      return_7f() ;
   }
   else
   {
      _chain_intr( int_10_original_vector ) ;
   }
}

/* ********************************************************************** */
/*                                                                        */
/*       Function     -   Interrupt routine int_2f()                      */
/*                                                                        */
/*       Description -    Internal INT 2fh interrupt handler              */
/*                                                                        */
/*                                                                        */
/* ********************************************************************** */
interrupt far int_2f( void )
{
   unsigned int     ax ;
   unsigned char    ah ;
   unsigned char    al ;

   ax = return_ax() ;
   ah = (unsigned char)(ax >> 8) ;
   al = (unsigned char)(ax & 0x0f);
   if( ah == 0x40 )
   {  /* Screen switch notification received */
      if (al == 0x01 )      /* Switch to background */
         PutXGAInVGA();
      else if (al == 0x02) /* Switch to foreground */
      {
         RedrawRequired = FALSE ;
         PutXGAIn132();
      }
   }
   if(int_2f_original_vector)   _chain_intr( int_2f_original_vector ) ;
}

/* ************************************************************************* */
/*                                                                           */
/*      Function     -   signal_handler()                                    */
/*                                                                           */
/*      Description -   error handler                                        */
/*                                                                           */
/*                                                                           */
/* ************************************************************************* */
void signal_handler( void )
{
   exit(0);
}


/* ************************************************************************* */
/*                                                                           */
/*      Function     -   exit_handler()                                      */
/*                                                                           */
/*      Description -   Exit handler                                         */
/*                                                                           */
/* ************************************************************************* */
int exit_handler( void )
{
   /* reset interrupt vectors to original values */
   if ( int_10_original_vector ) _dos_setvect( 0x10 , int_10_original_vector ) ;
   if ( int_2f_original_vector ) _dos_setvect( 0x2f , int_2f_original_vector ) ;

   if(XGAInVGA)PutXGAInVGA ();
   return (0) ;
}
```

```
/* ********************************************************************** */
/*                                                                        */
/*      Function    -   PutXGAIn132()                                     */
/*                                                                        */
/*      Description -   Puts the XGA into 132 mode                        */
/*                                                                        */
/* ********************************************************************** */
void PutXGAIn132( void )
{
   int ipdata ;
   int  far *bios ;

   outw ( IoRegBase + 0x0a , 0x1550 ) ;

   outw ( IoRegBase + 0x0a , 0x1450 ) ;

   outw ( IoRegBase + 0x0a , 0x0454 ) ;

   /* select scan lines for alphanumeric modes */
   inregs.h.ah = 0x12 ;
   inregs.h.al = 0x02 ;    /* 400 scan lines */
   inregs.h.bl = 0x30 ;
   int86( 0x10 , &inregs , &outregs ) ;

   /* set vga mode */
   inregs.h.ah = 0x00 ;
   inregs.h.al = 0x03 ;
   int86( 0x10 , &inregs , &outregs ) ;

   outp ( IoRegBase + 0x0a , 0x50 ) ;
   ipdata = inp  ( IoRegBase + 0x0b ) ;
   ipdata |= 0x01 ;
   outp ( IoRegBase + 0x0b , ipdata) ;

   outp ( IoRegBase + 0x0a , 0x50 ) ;
   ipdata = inp  ( IoRegBase + 0x0b ) ;
   ipdata &= 0xFD ;
   outp ( IoRegBase + 0x0b , ipdata) ;

   outp ( IoRegBase + 0x0a , 0x50 ) ;
   ipdata = inp  ( IoRegBase + 0x0b ) ;
   ipdata &= 0xFC ;
   outp ( IoRegBase + 0x0b , ipdata) ;

   outp ( IoRegBase  , 0x03 ) ;

   outpw( IoRegBase + 0x0a , 0x0154 ) ;

   outpw( IoRegBase + 0x0a , 0x8070 ) ;

   outp ( IoRegBase + 0x0a , 0x50 ) ;
   ipdata = inp  ( IoRegBase + 0x0b ) ;
   ipdata &= 0xef ;
   outp ( IoRegBase + 0x0b , ipdata) ;

   outp( 0x03d4 , 0x11 ) ;
   ipdata = inp( 0x03d5) ;
   ipdata &= 0x7f ;
```

```
    outp( 0x03d5 , ipdata );

    outp( 0x03d4 , 0x00 ) ;
    outp( 0x03d5 , 0xa4 );

    outp( 0x03d4 , 0x01 ) ;
    outp( 0x03d5 , 0x83 );

    outp( 0x03d4 , 0x02 ) ;
    outp( 0x03d5 , 0x84 );

    outp( 0x03d4 , 0x03 ) ;
    outp( 0x03d5 , 0x83 );

    outp( 0x03d4 , 0x04 ) ;
    outp( 0x03d5 , 0x90 );

    outp( 0x03d4 , 0x05 ) ;
    outp( 0x03d5 , 0x80 );

    outpw( IoRegBase + 0x0a , 0xa31a ) ;
    outpw( IoRegBase + 0x0a , 0x001b ) ;
    outpw( IoRegBase + 0x0a , 0x001c ) ;
    outpw( IoRegBase + 0x0a , 0x001e ) ;

    outp( 0x03d4 , 0x13 ) ;
    outp( 0x03d5 , 0x42 );

    outp( 0x03d4 , 0x11 ) ;
    ipdata = inp( 0x03d5) ;
    ipdata |= 0x80 ;
    outp( 0x03d5 , ipdata );

    outp ( IoRegBase + 0x0a , 0x50 ) ;
    ipdata = inp  ( IoRegBase + 0x0b ) ;
    ipdata |= 0x03 ;
    outp ( IoRegBase + 0x0b , ipdata) ;

    outp( 0x03c4 , 0x01 ) ;
    ipdata = inp( 0x03c5) ;
    ipdata |= 0x01 ;
    outp( 0x03c5 , ipdata );

    ipdata = inp( 0x03da) ;

    outp( 0x03c0 , 0x13 );

    outp( 0x03c0 , 0x00 );

    outp( 0x03c0 , 0x20 );

    bios = (int far *)0x40004a  ;
    *bios = 0x84 ;              /* tel BIOS we have 132 columns */

}
```

```
/* ********************************************************************** */
/*                                                                        */
/*      Function    -   PutXGAInVGA()                                     */
/*                                                                        */
/*      Description -   puts  the XGA into VGA mode                       */
/*                                                                        */
/* ********************************************************************** */
void PutXGAInVGA( void )
{
   int    i ;

   vram_address = (unsigned char far *)0xA0000000;

   /* clear 1st 256KB of VRAM */
   for (i = 0 ; i < 4 ; i++)
   {
      outp(IoRegBase + 8 , i);
      ptr = vram_address ;
      memset(ptr , 0 , 0x8000);  /* set 1st 32KB of 64KB aperture*/
      ptr = vram_address + 0x8000 ;
      memset(ptr , 0 , 0x8000);  /* set 2nd 32KB of 64KB aperture*/
   }

   for (i = 0 ; i < sizeof(vga_data) ; i = i + 3 )
   {
      if (vga_data[i+2])
         outpw( IoRegBase + vga_data[i] ,
                (((unsigned)vga_data[i+1]) << 8) + (unsigned)vga_data[i+2] ) ;
      else
         outp( IoRegBase + vga_data[i] , (int)vga_data[i+1] );
   }

   outp( 0x03c3 , 0x01 );


   /* select scan lines for alphanumeric modes */
   inregs.h.ah = 0x12 ;
   inregs.h.al = 0x02 ;    /* 400 scan lines */
   inregs.h.bl = 0x30 ;
   int86( 0x10 , &inregs , &outregs ) ;

   /* set vga mode */
   inregs.h.ah = 0x00 ;
   inregs.h.al = 0x03 ;
   int86( 0x10 , &inregs , &outregs ) ;

}
```

```
void DisplayText( void )
{
    int j ;
    char ch ;

    for (j=1 ; j < 24 ; j++ )
    {  /* sample code to fill screen with 24 x 132 chars */
    printf("This line ....................................") ;
    printf("..............................................") ;
    printf(".................  is 132 chars long") ;
    }
        printf("Press Enter to continue.......");
        ch = getchar();
}
```

Assembler Subroutines

See “Setting the XGA Subsystem into Extended Graphics Mode” on page A-2.

Index

Keyboards

Figures

Notes:

Introduction

A keyboard is used as an input device to the system. The input from the keyboard is used by different parts of the system. Depending on the function of the key, action can be taken by BIOS, the keyboard controller, an operating system, or an application. This section describes what happens as keystroke information passes through the system; from the keyboard, through the internal layers, and finally to an application.

Scan-Code Set Overview

Each keyboard supports more than one scan-code set. When a key is pressed by the user, the logic in the keyboard can generate a different scan code, depending on the scan-code set that is currently in use.

When attached to a PS/2* system, the keyboard has a defaults to scan-code set 2. This scan-code set is always enabled after the initial power-on of the keyboard or after keyboard reset. During the system POST (power-on self-test), some systems set the keyboard to to scan-code set 1. This setting will remain unless it is changed by the application or operating system. Most applications and operating systems only use scan-code set 1. If the system changes the scan-code set, the keyboard controller and BIOS translation routines are capable of interpreting the new scan-code set.

Keyboard Layouts

The characters on some keys are on both the top and front face. When there is more than one character on a keytop, use the key combinations in the following figure to produce the desired character.

Characters	Key Combinations
Lower-left character	Character key only
Upper-left character	Shift+character key
Lower-right character (or front face)	Alt Gr or Alt Car+character key
Upper-right character	Alt+Shift+character key

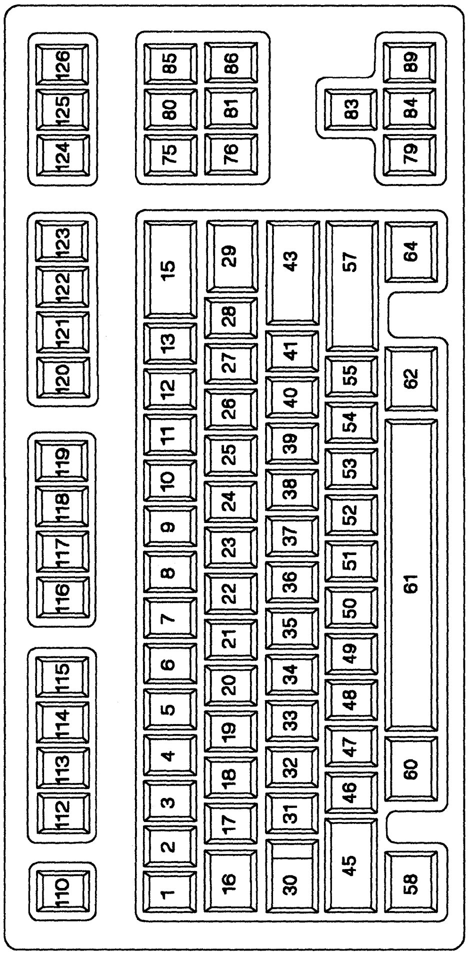
* PS/2 is a trademark of International Business Machines Corporation.

The keyboard layouts shown on the following pages are the key-position layouts and the United States layouts for each keyboard. They are:

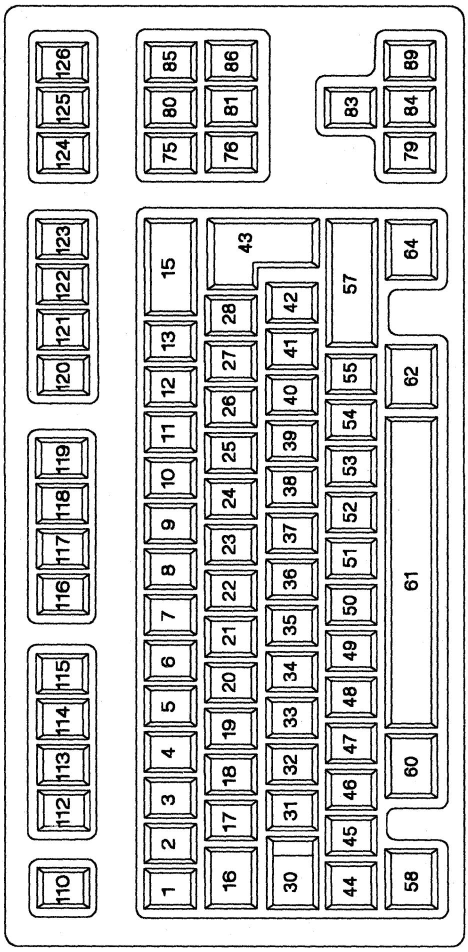
- Key-position layouts
  - 84-key position layout
  - 85-key position layout
  - 101-key position layout
  - 102-key position layout
  - 122-key position layout
- United States Keyboard Layouts
  - 84/85-key layout
  - 101/102-key layout
  - 122-key layout.

Non-U.S. layouts for each keyboard can be found in Appendix A.

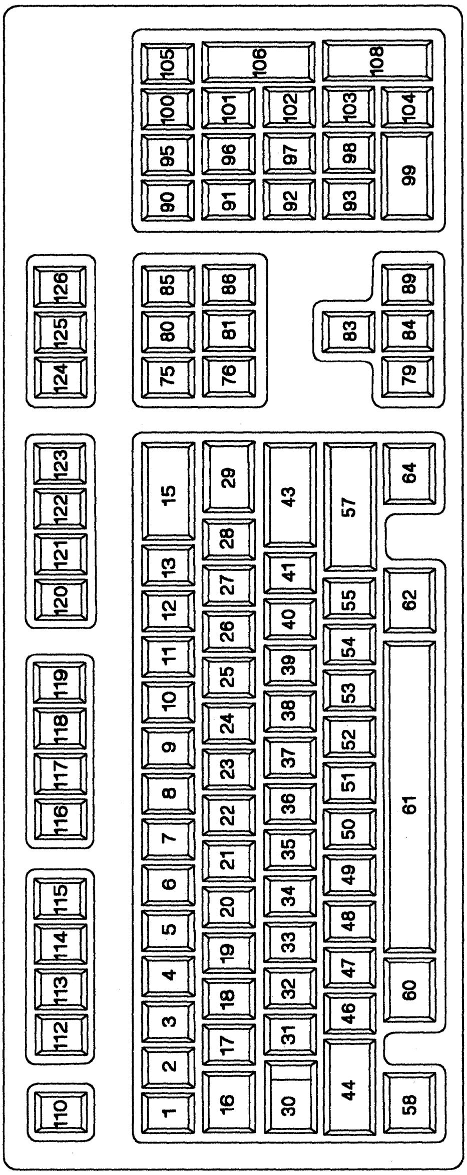
84-Key Position Layout



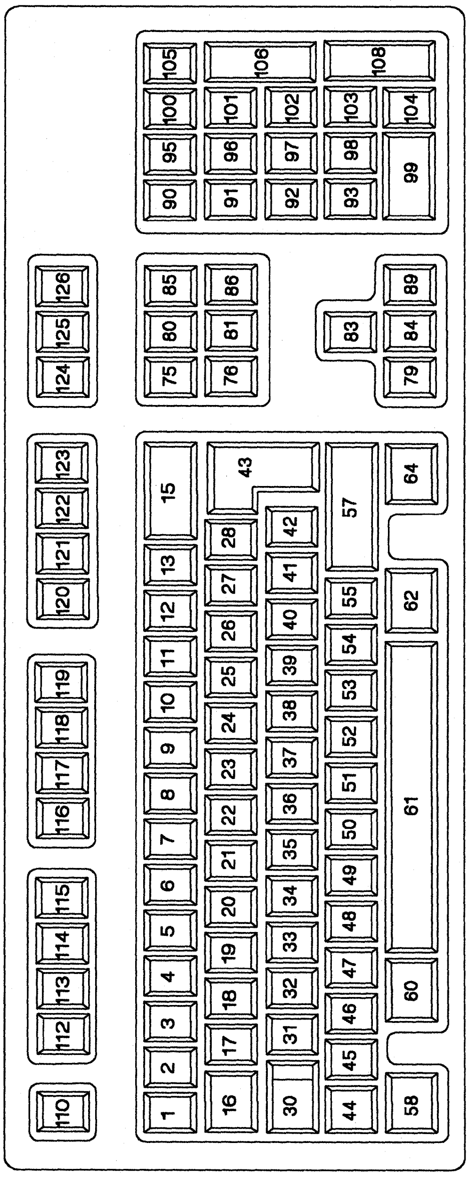
85-Key Position Layout



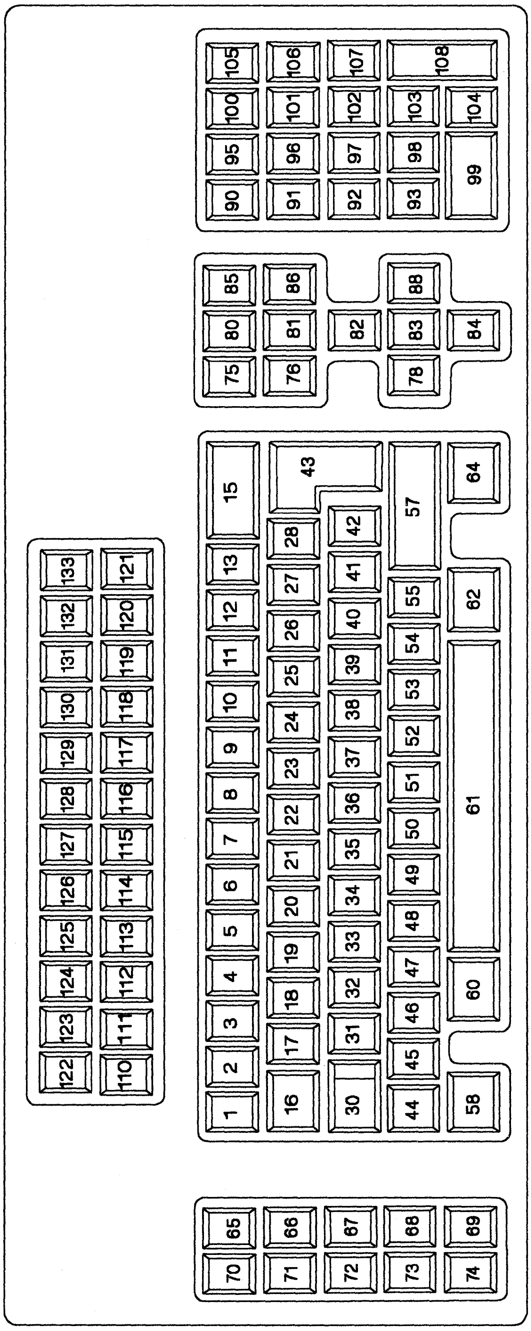
101-Key Position Layout



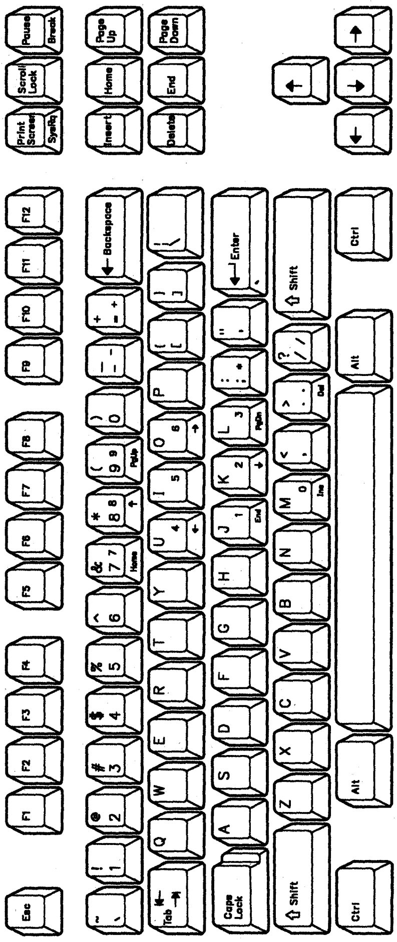
102-Key Position Layout



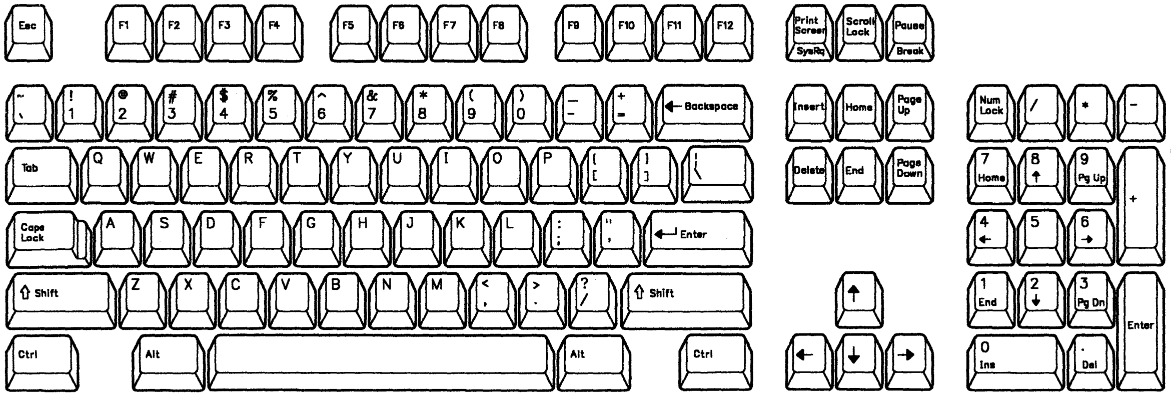
122-Key Position Layout

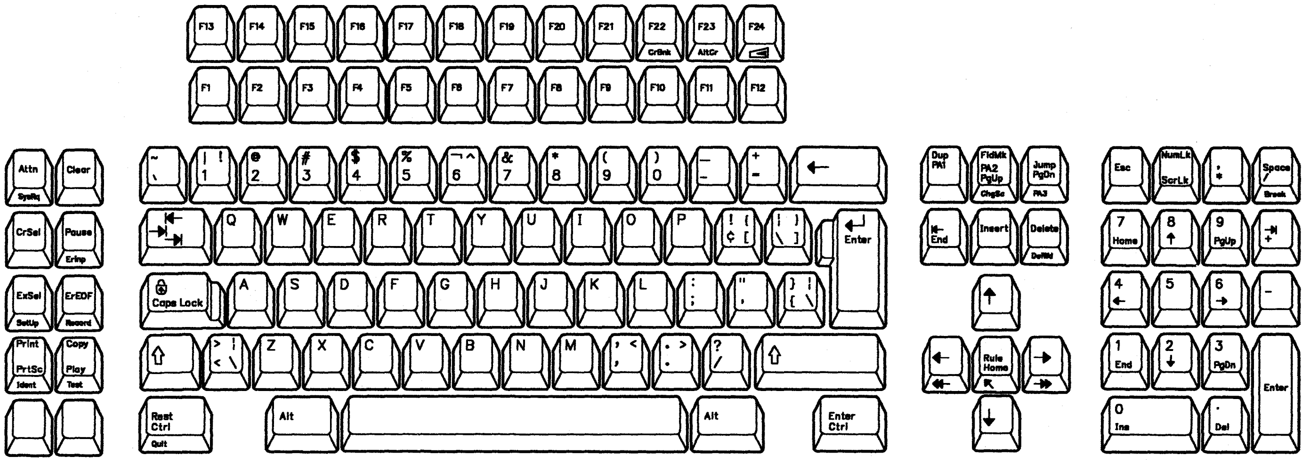


84-Key United States Keyboard



101-Key United States Keyboard




F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 CrBnk F23 AltCr F24
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Attn SysRq
Clear
CrSel
Pause ErInp
ExSel SetUp
ErEOF Record
Print PrtSc Ident
Copy Play Test
Caps Lock
Enter
Rest Ctrl Quit
Alt
Alt
Enter Ctrl
Dup PA1
FldMk PA2 PgUp ChgSc
Jump PgDn PA3
End
Insert
Delete DelWd
Rule Home
Esc
NumLk ScrLk
Space / Break
7 Home
8
9 PgUp
4
5
6
1 End
2
3 PgDn
0 Ins
. Del
Enter

Power-On Routine

The following activities take place when power is first applied to the keyboard.

Power-On Reset (POR)

The keyboard logic generates a 'power-on reset' signal when power is first applied to the keyboard. A POR takes a minimum of 150 milliseconds and a maximum of 2.0 seconds from the time power is first applied to the keyboard.

Basic Assurance Test

The basic assurance test (BAT) consists of a keyboard processor test, a checksum of the read-only memory (ROM), and a random-access memory (RAM) test. During the BAT, activity on the 'clock' and 'data' lines is ignored. The LEDs are turned on at the beginning and off at the end of the BAT. The BAT takes a minimum of 300 milliseconds and a maximum of 500 milliseconds. This is in addition to the time required by the POR.

On satisfactory completion of the BAT, a completion code (hex AA) is sent to the system, and keyboard scanning begins. If a BAT failure occurs, the keyboard sends an error code to the system. The keyboard is then disabled pending command input. Completion codes are sent between 450 milliseconds and 2.5 seconds after the POR, and between 300 and 500 milliseconds after a Reset command is acknowledged.

Immediately following a POR, the keyboard monitors the signals on the keyboard 'clock' and 'data' lines and sets the line protocol.

Commands from the System

The following figure shows the commands (and their hexadecimal values) that the system can send to the keyboard. All values not shown in the figure are reserved.

Command	Hex Value
Default Disable	F5
Echo	EE
Enable	F4
Invalid Command	EF
Invalid Command	F1
Read ID	F2
Resend	FE
Reset	FF
Select Alternate Scan Codes	F0
Set All Keys - Typematic	F7
- Make/Break	F8
- Make	F9
- Typematic/Make/Break	FA
Set Default	F6
Set Key Type - Typematic	FB
- Make/Break	FC
- Make	FD
Set/Reset Status Indicators	ED
Set Typematic Rate/Delay	F3

Figure 1. Keyboard Commands from the System

These commands can be sent to the keyboard at any time. The keyboard responds within 20 milliseconds, except when performing the BAT or executing a Reset command.

The following commands are in alphabetical order. They have different meanings when issued by the keyboard. See "Commands to the System" on page 18.

Default Disable (Hex F5): The Default Disable command resets all conditions to the power-on default state. The keyboard responds with ACK, clears its output buffer, sets the typematic rate/delay (and default key types in scan-code set 3 operation only), and clears the last typematic key. The keyboard stops scanning and awaits further instructions.

Echo (Hex EE): The Echo command is a diagnostic aid. When the keyboard receives this command, it issues a hex EE response. If previously enabled, it continues scanning.

Enable (Hex F4): When the Enable command is received, the keyboard responds with ACK, clears its output buffer, clears the last typematic key, and starts scanning.

Invalid Command (Hex EF and F1): Hex EF and hex F1 are invalid commands and are not supported. If one of these is sent, the keyboard does not acknowledge the command. It returns a Resend command and continues in its prior scanning state. No other activities occur.

Read ID (Hex F2): The Read ID command requests identification information from the keyboard. The keyboard responds with ACK, stops scanning, and sends the two keyboard ID bytes. The second byte must follow completion of the first by no more than 500 microseconds. After the output of the second ID byte, the keyboard resumes scanning. Keyboard IDs are listed in the following figure.

Keyboard Type	ID - Hex Value
84/85-key keyboard	84AB
101/102-key keyboard	83AB
122-key keyboard	86AB

Figure 2. Read ID Command Keyboard Identification

Resend (Hex FE): The Resend command is sent by the system when it detects an error in any transmission from the keyboard. It is sent only after a keyboard transmission and before the system allows the next keyboard output. When a Resend command is received, the keyboard sends the previous output again (unless the previous output was the Resend command, in which case the keyboard sends the last byte before the Resend command).

Reset (Hex FF): The Reset command is sent by the system to start a program reset and a keyboard internal self-test. The keyboard acknowledges the command with an ACK and ensures the system accepts ACK before executing the command. The system signals acceptance of ACK by raising the 'clock' and 'data' lines for a minimum of 500 microseconds. The keyboard is disabled from the time it receives the Reset command until ACK is accepted, or until another command is sent that overrides the previous command.

Following acceptance of ACK, the keyboard is reinitialized and performs the BAT. After returning the completion code, the keyboard defaults to scan-code set 2.

Select Alternate Scan Codes (Hex F0): The Select Alternate Scan Codes command instructs the keyboard to select one of three sets of scan codes. The keyboard acknowledges receipt of this command with ACK and clears both the output buffer and the typematic key (if one is active). The system then sends the option byte and the keyboard responds with another ACK. An option byte value of hex 01 selects scan-code set 1, hex 02 selects scan-code set 2, and hex 03 selects scan-code set 3.

An option byte value of hex 00 causes the keyboard to acknowledge with an ACK and send a byte telling the system which scan-code set is currently in use. To prevent the controller from translating this byte, disable the keyboard-controller translate mode.

After establishing the new scan-code set, the keyboard returns to the scanning state it was in before receiving the Select Alternate Scan Codes command.

Set All Keys (Hex F7, F8, F9, FA)

The Set All Keys commands instruct the keyboard to set all keys to a condition listed in the following figure.

Command	**Hex Value**
Set All Keys - Typematic	F7
Set All Keys - Make/Break	F8
Set All Keys - Make	F9
Set All Keys - Typematic/Make/Break	FA

Figure 3. Set All Keys Commands

The keyboard responds with ACK, clears its output buffer, sets all keys to the condition indicated by the command, and continues scanning (if it was previously enabled). Although these commands can be sent using any scan-code set, they affect only the operation of scan-code set 3.

Set Default (Hex F6): The Set Default command resets all conditions to the power-on default state. The keyboard responds with ACK, clears its output buffer, sets the default key types (scan-code set 3 operation only) and typematic rate/delay, clears the last typematic key, and continues scanning.

Set Key Type (Hex FB, FC, FD): The Set Key Type commands instruct the keyboard to set individual keys to a condition listed in the following figure.

Command	Hex Value
Set Key Type - Typematic	FB
Set Key Type - Make/Break	FC
Set Key Type - Make	FD

Figure 4. Set Key Type Commands

The keyboard responds with ACK, clears its output buffer, and prepares to receive key identification. The system identifies each key by its scan-code value, as defined in scan-code set 3. Only scan-code set 3 values are valid for key identification. The type of each identified key is set to the value indicated by the command.

These commands can be sent using any scan-code set, but affect only the operation of scan-code set 3.

Set/Reset Status Indicators (Hex ED): The Set/Reset Status Indicators command is sent by the system to set the indicators on the keyboard: Num Lock, Caps Lock, and Scroll Lock. The keyboard activates or deactivates these indicators when it receives a valid command-code sequence from the system. The command sequence begins with the command byte (hex ED). The keyboard responds with ACK, stops scanning, and waits for the option byte from the system. The bit assignments for this option byte are as follows.

Bit	Function
7 - 3	Reserved (must be 0's)
2	Caps Lock Indicator
1	Num Lock Indicator
0	Scroll Lock Indicator

Figure 5. Set/Reset Status Indicators

If a bit for an indicator is set to 1, the indicator is turned on. If a bit is set to 0, the indicator is turned off.

The keyboard responds to the option byte with ACK, sets the indicators and, if the keyboard was previously enabled, continues scanning. The state of the indicators reflects the bits in the option byte and can be activated or deactivated in any combination. If another command is received in place of the option byte, execution of the Set/Reset Mode Indicators command is stopped, with no change to the indicator states, and the new command is processed.

Immediately after power-on, the indicators default to the Off state. If the Set Default and Default Disable commands are received, the indicators remain in the state they were in before the command was received.

Set Typematic Rate/Delay (Hex F3): The system issues this command to change the typematic rate and delay. The keyboard responds to the command with ACK, stops scanning, and waits for the system to issue the rate/delay value byte. The keyboard responds to the rate/delay value byte with another ACK, sets the rate and delay to the values indicated, and continues scanning (if it was previously enabled). Bits 6 and 5 indicate the delay, and bits 4, 3, 2, 1, and 0 (the least-significant bit) indicate the rate. Bit 7, the most-significant bit, is always 0. The delay is determined by the following equation:

Delay = $(1 + A) \times 250$ milliseconds $\pm 20\%$.
where:
A = binary value of bits 6 and 5

The period (interval from one typematic output to the next) is determined by the following equation:

Period = $((8 + A) \times 2^{B})/2 \times 0.00834$ seconds $\pm 20\%$
where:
A = binary value of bits 2, 1, and 0
B = binary value of bits 4 and 3

The typematic rate (make codes per second) is 1 for each period.

Bit	Typematic Rate ± 20%	Bit	Typematic Rate ± 20%
00000	30.0	10000	7.5
00001	30.0	10001	6.7
00010	24.0	10010	6.0
00011	24.0	10011	5.5
00100	20.0	10100	5.0
00101	20.0	10101	4.6
00110	17.1	10110	4.3
00111	17.1	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

Figure 6. Typematic Rate

The default values for the system keyboard are as follows:

Typematic rate = 10.9 codes per second ± 20%

Delay = 500 milliseconds ± 20%.

The execution of this command stops without change to the existing rate if another command is received instead of the rate/delay value byte.

Commands to the System

The following figure shows the commands (and their hexadecimal values) that the keyboard can send to the system. All values not shown in the figure are reserved.

Hex Value	Command
Acknowledge (ACK)	FA
BAT Completion Code	AA
BAT Failure Code	FC
Break Code Prefix	F0
Echo	EE
Keyboard ID	83AB, 84AB, or 86AB (Dependent on keyboard type)
Key Detection Error/Overrun	00 (Code Sets 2 and 3)
Key Detection Error/Overrun	FF (Code Set 1)
Resend	FE

Figure 7. Keyboard Commands to the System

The commands the keyboard sends to the system are described in alphabetical order. They have different meanings when issued by the system. See "Commands from the System" on page 12.

Acknowledge (Hex FA): The Acknowledge command is a response to other commands. The keyboard issues ACK to any valid input other than an Echo or Resend command. If the keyboard is interrupted while sending ACK, it discards ACK and accepts and responds to the new command.

BAT Completion Code (Hex AA): The BAT Completion Code command is a command that indicates BAT status. Following satisfactory completion of the BAT, the keyboard sends hex AA. Any other code indicates a failure of the keyboard.

BAT Failure Code (Hex FC): The BAT Failure Code command is a command that indicates BAT status. If a BAT failure occurs, the keyboard sends this code, stops scanning, and waits for a system response or reset.

Break Code Prefix (Hex F0): The Break Code Prefix command is a command that transmits a code as the first byte of a 2-byte sequence to indicate the "break" of a key when scan-code set 2 or 3 is used. The keyboard will begin transmission of the second byte (typically within 2 milliseconds) when the interface is enabled after the output of the break code prefix.

Echo (Hex EE): The Echo command is a diagnostic aid. The keyboard sends this code in response to an Echo command.

Keyboard ID (Hex 83AB, 84AB or 86AB): The Keyboard ID command sends identification information from the keyboard. The keyboard responds to the Read ID command with ACK, stops scanning, and sends the two ID bytes. The low byte is sent first, followed by the high byte. Following the output of the keyboard ID, the keyboard begins scanning (if previously enabled). Keyboard IDs are listed in the following figure.

Keyboard Type	ID - Hex Value
84/85-key keyboard	84AB
101/102-key keyboard	83AB
122-key keyboard	86AB

Figure 8. Keyboard ID Command Keyboard Identification

Key Detection Error (Hex 00 or FF): The Key Detection Error command sends a key detection error character if conditions in the keyboard make it impossible to identify a switch closure. If the keyboard is using scan-code set 1, the code is hex FF. For sets 2 and 3, the code is hex 00.

Overrun (Hex 00 or FF): The Overrun command sends an overrun character that is placed in the keyboard buffer and replaces the last code when the buffer capacity has been exceeded. The code is sent to the system when it reaches the top of the buffer queue. If the keyboard is using scan-code set 1, the code is hex FF. For sets 2 and 3, the code is hex 00.

Resend (Hex FE): The Resend command is sent by the keyboard following receipt of an invalid input or any input with incorrect parity. If the system sends nothing to the keyboard, no response is required.

Sequential Key-Code Detection

The keyboard detects all the keys that are pressed and then sequentially sends each scan code. If the system is busy and is unable to accept the scan codes, the keyboard stores the scan codes in its internal buffer.

Although the keyboard can detect a key that is pressed before a previously pressed key is released, this ability should not be confused with system-supported two- and three-key combinations.

See “Two-Key Combinations” on page 22 and “Three-Key Combinations” on page 24. Each keystroke make-and-break combination is visible at the keyboard interface and Port 60H; however, a system-supported keystroke combination is translated by the system BIOS prior to INT 16H. All other keystroke combinations appear as individual keystrokes to the program at INT 16H.

To ensure future compatibility, application programs should not use any keystroke combination that is not listed in the figures in “Two-Key Combinations” on page 22 or “Three-Key Combinations” on page 24, even though the keyboard logic is able to detect the combinations on some keyboards.

Buffer

A first-in-first-out (FIFO) buffer in the keyboard stores the scan codes until the system is ready to receive them. A buffer-overrun condition occurs when the buffer is full and an additional key is pressed. When this happens, an overrun code is entered in the last position in the buffer. If more keys are pressed before the system allows keyboard output, the additional data is lost.

When the keyboard is allowed to send data, the bytes in the buffer are sent as in normal operation, and new data entered is detected and sent. Response codes do not occupy a buffer position.

If keystrokes generate a multiple-byte sequence, the entire sequence must fit into the available buffer space, or the keystroke is discarded and a buffer-overrun condition occurs.

Keys

There are three different key types; *make only, make/break,* and *typematic.*

Note: Scan-code set 3 allows key types to be changed by the system. See “101- and 102-Key Keyboard Scan-Code Set 3” on page 45 and “122-Key Keyboard Scan-Code Set 3” on page 48 for the default settings.

Make-only keys are keys that send out scan codes only when the key is pressed, not released. The Pause key and Space/Break key are examples of make only keys.

Make/break keys are keys that send out scan codes when the key is pressed. When the key is released, another scan code is sent out.

Except for the Pause key and the Space/Break key, all keys are make/break in scan-code sets 1 and 2.

Typematic keys are keys that send out scan codes when pressed and repeat these codes when the key is held down. Also, except for the Pause key and the Space/Break key, all keys are typematic in scan-code sets 1 and 2. When a key is pressed and held down, the keyboard sends the make code for that key, delays 500 milliseconds ±20%, and begins sending a make code for that key at a rate of 10.9 codes per second ±20%. The typematic rate and delay can be modified. See "Set Typematic Rate/Delay (Hex F3)" on page 16.

If two or more keys are held down, only the last key pressed repeats at the typematic rate. Typematic operation stops when the last key pressed is released, even if other keys are still held down. If a key is pressed and held down while keyboard transmission is inhibited, only the first make code is stored in the buffer. This prevents buffer overflow caused by typematic action.

Two-Key Combinations

All two-key combinations on the keyboard are supported with the left or right Shift key. In addition, the following figures show the other two-key combinations that are supported by the keyboard logic. *All other two-key combinations are reserved.* To ensure future compatibility, programs should ignore all reserved combinations.

In each two-key sequence shown in the following figures, the Alt and Ctrl combinations can use either the left or right Alt or Ctrl.

Alt Combinations		
Alt - Esc	Alt - -	Alt - h
Alt - F1	Alt - =	Alt - j
Alt - F2	Alt - Backsp	Alt - k
Alt - F3	Alt - Tab	Alt - l
Alt - F4	Alt - q	Alt - ;
Alt - F5	Alt - w	Alt - '
Alt - F6	Alt - e	Alt - z
Alt - F7	Alt - r	Alt - x
Alt - F8	Alt - t	Alt - c
Alt - F9	Alt - y	Alt - v
Alt - F10	Alt - u	Alt - b
Alt - F11	Alt - i	Alt - n
Alt - F12	Alt - o	Alt - m
Alt - `	Alt - p	Alt - ,
Alt - 1	Alt - [	Alt - .
Alt - 2	Alt -]	Alt - /
Alt - 3	Alt - \	Alt - Insert
Alt - 4	Alt - a	Alt - Delete
Alt - 5	Alt - s	Alt - Home
Alt - 6	Alt - d	Alt - End
Alt - 7	Alt - f	Alt - PgUp
Alt - 8	Alt - g	Alt - PgDn
Alt - 9		
Alt - 0		

Figure 9. Two-Key Combinations (Alt Combinations)

Ctrl Combinations		
Ctrl - Esc	Ctrl - -	Ctrl - h
Ctrl - F1	Ctrl - =	Ctrl - j
Ctrl - F2	Ctrl - Backsp	Ctrl - k
Ctrl - F3	Ctrl - Tab	Ctrl - l
Ctrl - F4	Ctrl - q	Ctrl - ;
Ctrl - F5	Ctrl - w	Ctrl - ’
Ctrl - F6	Ctrl - e	Ctrl - z
Ctrl - F7	Ctrl - r	Ctrl - x
Ctrl - F8	Ctrl - t	Ctrl - c
Ctrl - F9	Ctrl - y	Ctrl - v
Ctrl - F10	Ctrl - u	Ctrl - b
Ctrl - F11	Ctrl - i	Ctrl - n
Ctrl - F12	Ctrl - o	Ctrl - m
Ctrl - ‘	Ctrl - p	Ctrl - ,
Ctrl - 1	Ctrl - [	Ctrl - .
Ctrl - 2	Ctrl -]	Ctrl - /
Ctrl - 3	Ctrl - \	Ctrl - Insert
Ctrl - 4	Ctrl - a	Ctrl - Delete
Ctrl - 5	Ctrl - s	Ctrl - Home
Ctrl - 6	Ctrl - d	Ctrl - End
Ctrl - 7	Ctrl - f	Ctrl - PgUp
Ctrl - 8	Ctrl - g	Ctrl - PgDn
Ctrl - 9		
Ctrl - 0		

Figure 10. Two-Key Combinations (Ctrl Combinations)

Three-Key Combinations

The following figures show the other three-key combinations that are supported by the keyboard logic. *All other three-key combinations are reserved.* To ensure future compatibility, programs should ignore all reserved combinations.

In each three-key sequence shown in the following figures, the first two keys can be pressed in any order. Also, for Alt, Ctrl, and Shift, either the left or the right key can be pressed.

	Alt - Ctrl Combinations	
Alt - Ctrl - Esc	Alt - Ctrl - -	Alt - Ctrl - h
Alt - Ctrl - F1	Alt - Ctrl - =	Alt - Ctrl - j
Alt - Ctrl - F2	Alt - Ctrl - Backsp	Alt - Ctrl - k
Alt - Ctrl - F3	Alt - Ctrl - Tab	Alt - Ctrl - l
Alt - Ctrl - F4	Alt - Ctrl - q	Alt - Ctrl - ;
Alt - Ctrl - F5	Alt - Ctrl - w	Alt - Ctrl - '
Alt - Ctrl - F6	Alt - Ctrl - e	Alt - Ctrl - z
Alt - Ctrl - F7	Alt - Ctrl - r	Alt - Ctrl - x
Alt - Ctrl - F8	Alt - Ctrl - t	Alt - Ctrl - c
Alt - Ctrl - F9	Alt - Ctrl - y	Alt - Ctrl - v
Alt - Ctrl - F10	Alt - Ctrl - u	Alt - Ctrl - b
Alt - Ctrl - F11	Alt - Ctrl - i	Alt - Ctrl - n
Alt - Ctrl - F12	Alt - Ctrl - o	Alt - Ctrl - m
Alt - Ctrl - `	Alt - Ctrl - p	Alt - Ctrl - ,
Alt - Ctrl - 1	Alt - Ctrl - [	Alt - Ctrl - .
Alt - Ctrl - 2	Alt - Ctrl -]	Alt - Ctrl - /
Alt - Ctrl - 3	Alt - Ctrl - \	Alt - Ctrl - Insert
Alt - Ctrl - 4	Alt - Ctrl - a	Alt - Ctrl - Delete
Alt - Ctrl - 5	Alt - Ctrl - s	Alt - Ctrl - Home
Alt - Ctrl - 6	Alt - Ctrl - d	Alt - Ctrl - End
Alt - Ctrl - 7	Alt - Ctrl - f	Alt - Ctrl - PgUp
Alt - Ctrl - 8	Alt - Ctrl - g	Alt - Ctrl - PgDn
Alt - Ctrl - 9		
Alt - Ctrl - 0		

Figure 11. Three-Key Combinations (Alt - Ctrl Combinations)

	Alt - Shift Combinations	
Alt - Shift - Esc	Alt - Shift - -	Alt - Shift - h
Alt - Shift - F1	Alt - Shift - =	Alt - Shift - j
Alt - Shift - F2	Alt - Shift - Backsp	Alt - Shift - k
Alt - Shift - F3	Alt - Shift - Tab	Alt - Shift - l
Alt - Shift - F4	Alt - Shift - q	Alt - Shift - ;
Alt - Shift - F5	Alt - Shift - w	Alt - Shift - ′
Alt - Shift - F6	Alt - Shift - e	Alt - Shift - z
Alt - Shift - F7	Alt - Shift - r	Alt - Shift - x
Alt - Shift - F8	Alt - Shift - t	Alt - Shift - c
Alt - Shift - F9	Alt - Shift - y	Alt - Shift - v
Alt - Shift - F10	Alt - Shift - u	Alt - Shift - b
Alt - Shift - F11	Alt - Shift - i	Alt - Shift - n
Alt - Shift - F12	Alt - Shift - o	Alt - Shift - m
Alt - Shift - ‘	Alt - Shift - p	Alt - Shift - ,
Alt - Shift - 1	Alt - Shift - [	Alt - Shift - .
Alt - Shift - 2	Alt - Shift -]	Alt - Shift - /
Alt - Shift - 3	Alt - Shift - \	Alt - Shift - Insert
Alt - Shift - 4	Alt - Shift - a	Alt - Shift - Delete
Alt - Shift - 5	Alt - Shift - s	Alt - Shift - Home
Alt - Shift - 6	Alt - Shift - d	Alt - Shift - End
Alt - Shift - 7	Alt - Shift - f	Alt - Shift - PgUp
Alt - Shift - 8	Alt - Shift - g	Alt - Shift - PgDn
Alt - Shift - 9		
Alt - Shift - 0		

Figure 12. Three-Key Combinations (Alt - Shift Combinations)

	Ctrl - Shift Combinations	
Ctrl - Shift - Esc	Ctrl - Shift - -	Ctrl - Shift - h
Ctrl - Shift - F1	Ctrl - Shift - =	Ctrl - Shift - j
Ctrl - Shift - F2	Ctrl - Shift - Backsp	Ctrl - Shift - k
Ctrl - Shift - F3	Ctrl - Shift - Tab	Ctrl - Shift - l
Ctrl - Shift - F4	Ctrl - Shift - q	Ctrl - Shift - ;
Ctrl - Shift - F5	Ctrl - Shift - w	Ctrl - Shift - '
Ctrl - Shift - F6	Ctrl - Shift - e	Ctrl - Shift - z
Ctrl - Shift - F7	Ctrl - Shift - r	Ctrl - Shift - x
Ctrl - Shift - F8	Ctrl - Shift - t	Ctrl - Shift - c
Ctrl - Shift - F9	Ctrl - Shift - y	Ctrl - Shift - v
Ctrl - Shift - F10	Ctrl - Shift - u	Ctrl - Shift - b
Ctrl - Shift - F11	Ctrl - Shift - i	Ctrl - Shift - n
Ctrl - Shift - F12	Ctrl - Shift - o	Ctrl - Shift - m
Ctrl - Shift - ‘	Ctrl - Shift - p	Ctrl - Shift - ,
Ctrl - Shift - 1	Ctrl - Shift - [	Ctrl - Shift - .
Ctrl - Shift - 2	Ctrl - Shift -]	Ctrl - Shift - /
Ctrl - Shift - 3	Ctrl - Shift - \	Ctrl - Shift - Insert
Ctrl - Shift - 4	Ctrl - Shift - a	Ctrl - Shift - Delete
Ctrl - Shift - 5	Ctrl - Shift - s	Ctrl - Shift - Home
Ctrl - Shift - 6	Ctrl - Shift - d	Ctrl - Shift - End
Ctrl - Shift - 7	Ctrl - Shift - f	Ctrl - Shift - PgUp
Ctrl - Shift - 8	Ctrl - Shift - g	Ctrl - Shift - PgDn
Ctrl - Shift - 9		
Ctrl - Shift - 0		

Figure 13. Three-Key Combinations (Ctrl - Shift Combinations)

Scan Codes

The following figures show the hexadecimal values for the keyboard's scan-code sets 1, 2, and 3. The system defaults to scan-code set 2, but can be switched to set 1 or set 3. See "Select Alternate Scan Codes (Hex F0)" on page 14.

Using the Scan-Code Tables

For scan-code sets 1 and 2, the first and second columns are the key numbers and the actual key associated with each key number. The next five columns show the base scan code and the scan codes generated when the various shift keys (Alt, Ctrl, and Shift) are simultaneously pressed along with the base key, as well as taking into consideration the state of the Num Lock key.

For scan-code set 3, the first and second columns are the key numbers and the actual key associated with each key number. The next column shows the base scan code generated in scan-code set 3. The last column shows the default key type.

Note: To determine scan codes for the 84/85-key keyboard, the 101/102-key keyboard scan-code tables can be used with certain exceptions. These exceptions are shown following each 101/102-key keyboard scan-code table on pages 33 and 40. Scan-code set 3 for the 84/85-key keyboard is the same as the 101/102-key keyboard scan-code set 3.

Typematic Keys

The typematic scan code is always equal to the base-case scan code of the key pressed, regardless of any shift keys which may be pressed. For example, using the figures for scan-code set 1, 101- and 102-key keyboard, the scan code generated for key #75 (Insert) for the left shift case is E0 AA E0 52. If the key were held down past the typematic delay time, the next scan codes sent would be E0 52 (the base scan codes), which will be repeated until the Insert key is released, or until any other key is pressed.

For scan-code sets 1 and 2 on the 101- and 102-key keyboard, all keys are typematic except key #126 (Pause) which is a make-only key. For scan-code set 3, key types are shown in the last column.

For scan-code sets 1 and 2 on the 122-key keyboard, all keys are typematic except key #66 (Pause) and key #105 (Space/Break) which are make-only keys. For scan-code set 3, key types are shown in the last column.

Base Case Column

The Base Case column of the scan-code tables shows the make-and-break codes sent for each key. The make code is sent when the key is pressed and the break code is sent when the key is released. The make codes and the break codes are separated in the table by a slash ("/"). For example, using the figures for scan-code set 1, 101- and 102-key keyboard, the make code for key #15 (Backspace) is 0E and the break code is 8E.

The Base Case column also should be used when the base key is pressed along with any shift key (left Shift, right Shift, or both) when Num Lock is in effect. Num Lock cancels the effect of the shift keys. The Base Case column should be used in the following cases:

- Base key only
- Left Shift + Num Lock + base key
- Right Shift + Num Lock + base key
- Left Shift + right Shift + Num Lock + base key.

Shift Case Column

The Shift Case column of the scan-code tables shows the make-and-break codes sent when the base key is pressed (or released) while the left Shift is held down. This column also should be used when the base key is pressed along with left Shift and Alt key, left Shift and Ctrl key, or left Shift, Alt and Ctrl. The Shift Case column (left Shift) should be used in the following cases:

- Left Shift + base key
- Left Shift + Alt + base key
- Left Shift + Ctrl + base key
- Left Shift + Alt + Ctrl + base key.

If the right Shift is pressed, the scan codes sent are similar to the left shift case, with the following exceptions: The make and break codes for the left shift are replaced with the make and break codes for the right shift. For example, using the figures for scan-code set 1, 101- and 102-key keyboard, the scan code for key #75 (Insert) for the left shift case is:

```
E0 AA E0 52  (make)
E0 D2 E0 2A  (break)
```

The right shift case is:

```
E0 B6 E0 52  (make)
E0 D2 E0 36  (break)
```

If both left and right shift keys are pressed:

```
E0 AA E0 B6 E0 52  (make)
E0 D2 E0 36 E0 2A  (break)
```

Num Lock Case Column

The Num Lock case column of the scan-code tables shows the make and break codes sent when the base key is pressed (or released) while the Num Lock indicator is on. If the Alt key or the Ctrl key is pressed, use the corresponding column. If the Shift key is pressed, use the Base Case Column.

Alt Case Column
The Alt Case column of the scan-code tables shows the make-and-break codes sent when the base key is pressed (or released) while either the left Alt or the right Alt is held down. This column also should be used when the base key is pressed along with Alt and Ctrl, Alt while the Num Lock indicator is on, or Alt and Ctrl while the Num Lock indicator is on. The Alt Case column should be used in the following cases:

- Alt + base key
- Alt + Ctrl + base key
- Alt + Num Lock + base key
- Alt + Ctrl + Num Lock + base key.

Ctrl Case Column
The Ctrl Case column of the scan-code tables shows the make-and-break codes sent when the base key is pressed (or released) while either the left Ctrl or the right Ctrl is held down. This column also should be used when the base key is pressed along with either Ctrl while the Num Lock indicator is on. The Ctrl Case column should be used in the following cases:

- Ctrl + base key
- Ctrl + Num Lock + base key.

101- and 102-Key Keyboard Scan-Code Set 1

Key #	101-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
1	'	29/A9	**	**	**	**
2	1	02/82	**	**	**	**
3	2	03/83	**	**	**	**
4	3	04/84	**	**	**	**
5	4	05/85	**	**	**	**
6	5	06/86	**	**	**	**
7	6	07/87	**	**	**	**
8	7	08/88	**	**	**	**
9	8	09/89	**	**	**	**
10	9	0A/8A	**	**	**	**
11	0	0B/8B	**	**	**	**
12	-	0C/8C	**	**	**	**
13	=	0D/8D	**	**	**	**
15	Backspace	0E/8E	**	**	**	**
16	Tab	0F/8F	**	**	**	**
17	Q	10/90	**	**	**	**
18	W	11/91	**	**	**	**
19	E	12/92	**	**	**	**
20	R	13/93	**	**	**	**
21	T	14/94	**	**	**	**
22	Y	15/95	**	**	**	**
23	U	16/96	**	**	**	**
24	I	17/97	**	**	**	**
25	O	18/98	**	**	**	**
26	P	19/99	**	**	**	**
27	[	1A/9A	**	**	**	**
28	]	1B/9B	**	**	**	**
29	\	2B/AB	**	**	**	**
30	Caps Lock	3A/BA	**	**	**	**
31	A	1E/9E	**	**	**	**
32	S	1F/9F	**	**	**	**
33	D	20/A0	**	**	**	**
34	F	21/A1	**	**	**	**
35	G	22/A2	**	**	**	**
36	H	23/A3	**	**	**	**
37	J	24/A4	**	**	**	**
38	K	25/A5	**	**	**	**
39	L	26/A6	**	**	**	**
40	;	27/A7	**	**	**	**
41	'	28/A8	**	**	**	**
42	(No Key****)	2B/AB	**	**	**	**
43	Enter	1C/9C	**	**	**	**
44	L. Shift*	2A/AA	**	**	**	**
45	(No Key****)	56/D6	**	**	**	**

Figure 14 (Part 1 of 3). 101- and 102-Key Keyboard Scan-Code Set 1

Key #	101-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
46	Z	2C/AC	**	**	**	**
47	X	2D/AD	**	**	**	**
48	C	2E/AE	**	**	**	**
49	V	2F/AF	**	**	**	**
50	B	30/B0	**	**	**	**
51	N	31/B1	**	**	**	**
52	M	32/B2	**	**	**	**
53	,	33/B3	**	**	**	**
54	.	34/B4	**	**	**	**
55	/	35/B5	**	**	**	**
57	R. Shift*	36/B6	**	**	**	**
58	L. Ctrl*	1D/9D	**	**	**	**
60	L. Alt*	38/B8	**	**	**	**
61	Space Bar	39/B9	**	**	**	**
62	R. Alt*	E0 38/E0 B8	**	**	**	**
64	R. Ctrl*	E0 1D/E0 9D	**	**	**	**
75	Insert	E0 52/E0 D2	E0 AA E0 52/ E0 D2 E0 2A	E0 2A E0 52/ E0 D2 E0 AA	**	**
76	Delete	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	Home	E0 47/E0 C7	E0 AA E0 47/ E0 C7 E0 2A	E0 2A E0 47/ E0 C7 E0 AA	**	**
81	End	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	Page Up	E0 49/E0 C9	E0 AA E0 49/ E0 C9 E0 2A	E0 2A E0 49/ E0 C9 E0 AA	**	**
86	Page Down	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	**	**
90	Num Lock*	45/C5	**	**	**	**

Figure 14 (Part 2 of 3). 101- and 102-Key Keyboard Scan-Code Set 1

Key #	101-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
91	K.P. Home	47/C7	**	**	**	**
92	K.P. ←	4B/CB	**	**	**	**
93	K.P. End	4F/CF	**	**	**	**
95	/	E0 35/E0 B5	E0 AA E0 35/ E0 B5 E0 2A	**	**	**
96	K.P. ↑	48/C8	**	**	**	**
97	K.P. 5	4C/CC	**	**	**	**
98	K.P. ↓	50/D0	**	**	**	**
99	K.P. Ins	52/D2	**	**	**	**
100	K.P. *	37/B7	**	**	**	**
101	K.P. Pgup	49/C9	**	**	**	**
102	K.P. →	4D/CD	**	**	**	**
103	K.P. Pgdn	51/D1	**	**	**	**
104	K.P. Del	53/D3	**	**	**	**
105	K.P. -	4A/CA	**	**	**	**
106	K.P. +	4E/CE	**	**	**	**
108	K.P. Enter	E0 1C/E0 9C	**	**	**	**
110	Esc	01/81	**	**	**	**
112	F1	3B/BB	**	**	**	**
113	F2	3C/BC	**	**	**	**
114	F3	3D/BD	**	**	**	**
115	F4	3E/BE	**	**	**	**
116	F5	3F/BF	**	**	**	**
117	F6	40/C0	**	**	**	**
118	F7	41/C1	**	**	**	**
119	F8	42/C2	**	**	**	**
120	F9	43/C3	**	**	**	**
121	F10	44/C4	**	**	**	**
122	F11	57/D7	**	**	**	**
123	F12	58/D8	**	**	**	**
124	Print Scr/SysReq	E0 2A E0 37/ E0 B7 E0 AA	E0 37/E0 B7 (see ***)	**	54/D4 (see ***)	E0 37/E0 B7 (see ***)
125	Scrl Lock	46/C6	**	**	**	**
126	Pause/ Break	E1 1D 45 E1 9D C5	**	**	**	E0 46 E0 C6

* = May cause case change (See Column Heading to determine proper case).
** = Same as base case.
*** = Num Lock does not cancel L. Shift or R. Shift, L. Shift and R. Shift do not change Alt or Ctrl cases.
**** = Key does not exist in 101-key keyboard, but does exist in 102-key keyboard.

Figure 14 (Part 3 of 3). 101- and 102-Key Keyboard Scan-Code Set 1

84- and 85-Key Keyboard Scan-Code Set 1

Key #	84-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
8	7/7	08/88	**	47/C7	**	**
9	8/8	09/89	**	48/C8	**	**
10	9/9	0A/8A	**	49/C9	**	**
12	-/-	0C/8C	**	4A/CA	**	**
13	=/+	0D/8D	**	4E/CE	**	**
23	U/4	16/96	**	4B/CB	**	**
24	I/5	17/97	**	4C/CC	**	**
25	O/6	18/98	**	4D/CD	**	**
37	J/1	24/A4	**	4F/CF	**	**
38	K/2	25/A5	**	50/D0	**	**
39	L/3	26/A6	**	51/D1	**	**
40	;/*	27/A7	**	37/B7	**	**
43	Enter	1C/9C	**	E0 1C/E0 9C	**	**
52	M/0	32/B2	**	52/D2	**	**
54	./.	34/B4	**	53/D3	**	**
55	/	35/B5	E0 AA E0 35/ E0 B5 E0 2A (With Num Lock on)	E0 35/E0 B5	**	**
125	Scrl Lock	46/C6	45/C5 *	**	**	**

* = Key 125 becomes Num Lock in the shift case
** = Same as base case

Figure 15. 84- and 85-Key Keyboard Scan-Code Set 1. These are exceptions to the 101/102-key keyboard.

122-Key Keyboard Scan-Code Set 1

Key #	122-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
1	'	29/A9	**	**	**	**
2	1	02/82	**	**	**	**
3	2	03/83	**	**	**	**
4	3	04/84	**	**	**	**
5	4	05/85	**	**	**	**
6	5	06/86	**	**	**	**
7	6	07/87	**	**	**	**
8	7	08/88	**	**	**	**
9	8	09/89	**	**	**	**
10	9	0A/8A	**	**	**	**
11	0	0B/8B	**	**	**	**
12	-	0C/8C	**	**	**	**
13	=	0D/8D	**	**	**	**
15	Backspace	0E/8E	**	**	**	**
16	Tab	0F/8F	**	**	**	**
17	Q	10/90	**	**	**	**
18	W	11/91	**	**	**	**
19	E	12/92	**	**	**	**
20	R	13/93	**	**	**	**
21	T	14/94	**	**	**	**
22	Y	15/95	**	**	**	**
23	U	16/96	**	**	**	**
24	I	17/97	**	**	**	**
25	O	18/98	**	**	**	**
26	P	19/99	**	**	**	**
27	[	1A/9A	**	**	**	**
28	]	1B/9B	**	**	**	**
30	Caps Lock	3A/BA	**	**	**	**
31	A	1E/9E	**	**	**	**
32	S	1F/9F	**	**	**	**
33	D	20/A0	**	**	**	**
34	F	21/A1	**	**	**	**
35	G	22/A2	**	**	**	**
36	H	23/A3	**	**	**	**
37	J	24/A4	**	**	**	**
38	K	25/A5	**	**	**	**
39	L	26/A6	**	**	**	**
40	;	27/A7	**	**	**	**
41	'	28/A8	**	**	**	**
42	\	2B/AB	**	**	**	**
43	Enter	1C/9C	**	**	**	**
44	L. Shift*	2A/AA	**	**	**	**
45	\	56/D6	**	**	**	**
46	Z	2C/AC	**	**	**	**
47	X	2D/AD	**	**	**	**

Figure 16 (Part 1 of 3). 122-Key Keyboard Scan-Code Set 1

Key #	122-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
48	C	2E/AE	**	**	**	**
49	V	2F/AF	**	**	**	**
50	B	30/B0	**	**	**	**
51	N	31/B1	**	**	**	**
52	M	32/B2	**	**	**	**
53	,	33/B3	**	**	**	**
54	.	34/B4	**	**	**	**
55	/	35/B5	**	**	**	**
57	R. Shift*	36/B6	**	**	**	**
58	L. Ctrl*	1D/9D	**	**	**	**
60	L. Alt*	38/B8	**	**	**	**
61	Space Bar	39/B9	**	**	**	**
62	R. Alt*	E0 38/E0 B8	**	**	**	**
64	R. Ctrl*	E0 1D/E0 9D	**	**	**	**
65	Clear	76/F6	**	**	**	**
66	Pause	E1 1D 45 E1 9D C5	**	**	**	E1 45 E1 C5
67	ErEOF	6D/ED	**	**	**	**
68	Copy/Play	6F/EF	**	**	**	**
69	(No Key)	6C/EC	**	**	**	**
70	Attn/Sysreq	71/F1	**	**	54/D4	**
71	CrSel	72/F2	**	**	**	**
72	ExSel	74/F4	**	**	**	**
73	PrtSc	E0 2A E0 37/ E0 B7 E0 AA	E0 37/E0 B7 (see note***)	**	E0 37/E0 B7 (see note***)	E0 37/E0 B7 (see note***
74	(No Key)	75/F5	**	**	**	**
75	DupPA1	5A/DA	**	**	**	**
76	End	E0 4F/E0 CF	E0 AA E0 4F/ E0 CF E0 2A	E0 2A E0 4F/ E0 CF E0 AA	**	**
78	←	E0 4B/E0 CB	E0 AA E0 4B/ E0 CB E0 2A	E0 2A E0 4B/ E0 CB E0 AA	**	**
80	PgUp	E0 49/E0 C9	E0 AA E0 49/ E0 C9 E0 2A	E0 2A E0 49/ E0 C9 E0 AA	**	
81	Ins	E0 52/E0 D2	E0 AA E0 52/ E0 D2 E0 2A	E0 2A E0 52/ E0 D2 E0 AA	**	**
82	↑	E0 48/E0 C8	E0 AA E0 48/ E0 C8 E0 2A	E0 2A E0 48/ E0 C8 E0 AA	**	**
83	Home	E0 47/E0 C7	E0 AA E0 47/ E0 C7 E0 2A	E0 2A E0 47/ E0 C7 E0 AA	**	**
84	↓	E0 50/E0 D0	E0 AA E0 50/ E0 D0 E0 2A	E0 2A E0 50/ E0 D0 E0 AA	**	**
85	PgDn	E0 51/E0 D1	E0 AA E0 51/ E0 D1 E0 2A	E0 2A E0 51/ E0 D1 E0 AA	**	**
86	Del	E0 53/E0 D3	E0 AA E0 53/ E0 D3 E0 2A	E0 2A E0 53/ E0 D3 E0 AA	**	**
88	→	E0 4D/E0 CD	E0 AA E0 4D/ E0 CD E0 2A	E0 2A E0 4D/ E0 CD E0 AA	**	**

Figure 16 (Part 2 of 3). 122-Key Keyboard Scan-Code Set 1

Key #	122-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
90	Esc	01/81	**	**	**	**
91	K.P. Home/7	47/C7	**	**	**	**
92	K.P. ←/4	4B/CB	**	**	**	**
93	K.P. End/1	4F/CF		**	**	**
95	ScrLk/NumL	46/C6	45/C5	46/C6	46/C6	46/C6
96	K.P. ↑/8	48/C8	**	**	**	**
97	K.P. 5	4C/CC	**	**	**	**
98	K.P. ↓/2	50/D0	**	**	**	**
99	K.P. Ins/0	52/D2	**	**	**	**
100	K.P. *	37/B7	**	**	**	**
101	K.P. PgUp/9	49/C9	**	**	**	**
102	K.P. →/6	4D/CD	**	**	**	**
103	K.P. PgDn/	51/D1	**	**	**	**
104	K.P. Del/.	53/D3	**	**	**	**
105	Space/Break	E0 39/E0 B9	**	**	**	E0 46/E0 C6
106	K.P. +	4E/CE	**	**	**	**
107	K.P. -	4A/CA				
108	K.P. Enter	E0 1C/E0 9C	**	**	**	**
110	F1	3B/BB	**	**	**	**
111	F2	3C/BC	**	**	**	**
112	F3	3D/BD	**	**	**	**
113	F4	3E/BE	**	**	**	**
114	F5	3F/BF	**	**	**	**
115	F6	40/C0	**	**	**	**
116	F7	41/C1	**	**	**	**
117	F8	42/C2	**	**	**	**
118	F9	43/C3	**	**	**	**
119	F10	44/C4	**	**	**	**
120	F11	57/D7	**	**	**	**
121	F12	58/D8	**	**	**	**
122	F13	5B/DB	**	**	**	**
123	F14	5C/DC	**	**	**	**
124	F15	5D/DD	**	**	**	**
125	F16	63/E3	**	**	**	**
126	F17	64/E4	**	**	**	**
127	F18	65/E5	**	**	**	**
128	F19	66/E6	**	**	**	**
129	F20	67/E7	**	**	**	**
130	F21	68/E8	**	**	**	**
131	F22	69/E9	**	**	**	**
132	F23	6A/EA	**	**	**	**
133	F24	6B/EB	**	**	**	**

* = May cause case change (See Column Heading to determine proper case)
** = Same as base case
***= Num Lock does not cancel L. Shift or R. Shift, L. Shift and R. Shift do not change Alt or Ctrl cases

Figure 16 (Part 3 of 3). 122-Key Keyboard Scan-Code Set 1

101- and 102-Key Keyboard Scan-Code Set 2

Key #	101-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
1	'	0E/F0 0E	**	**	**	**
2	1	16/F0 16	**	**	**	**
3	2	1E/F0 1E	**	**	**	**
4	3	26/F0 26	**	**	**	**
5	4	25/F0 25	**	**	**	**
6	5	2E/F0 2E	**	**	**	**
7	6	36/F0 36	**	**	**	**
8	7	3D/F0 3D	**	**	**	**
9	8	3E/F0 3E	**	**	**	**
10	9	46/F0 46	**	**	**	**
11	0	45/F0 45	**	**	**	**
12	-	4E/F0 4E	**	**	**	**
13	=	55/F0 55	**	**	**	**
15	Backspace	66/F0 66	**	**	**	**
16	Tab	0D/F0 0D	**	**	**	**
17	Q	15/F0 15	**	**	**	**
18	W	1D/F0 1D	**	**	**	**
19	E	24/F0 24	**	**	**	**
20	R	2D/F0 2D	**	**	**	**
21	T	2C/F0 2C	**	**	**	**
22	Y	35/F0 35	**	**	**	**
23	U	3C/F0 3C	**	**	**	**
24	I	43/F0 43	**	**	**	**
25	O	44/F0 44	**	**	**	**
26	P	4D/F0 4D	**	**	**	**
27	[	54/F0 54	**	**	**	**
28	]	5B/F0 5B	**	**	**	**
29	\	5D/F0 5D	**	**	**	**
30	Caps Lock	58/F0 58	**	**	**	**
31	A	1C/F0 1C	**	**	**	**
32	S	1B/F0 1B	**	**	**	**
33	D	23/F0 23	**	**	**	**
34	F	2B/F0 2B	**	**	**	**
35	G	34/F0 34	**	**	**	**
36	H	33/F0 33	**	**	**	**
37	J	3B/F0 3B	**	**	**	**
38	K	42/F0 42	**	**	**	**
39	L	4B/F0 4B	**	**	**	**
40	;	4C/F0 4C	**	**	**	**
41	'	52/F0 52	**	**	**	**
42	(No Key****)	5D/F0 5D	**	**	**	**
43	Enter	5A/F0 5A	**	**	**	**
44	L. Shift*	12/F0 12	**	**	**	**
45	(No Key****)	61/F0 61	**	**	**	**
46	Z	1A/F0 1A	**	**	**	**

Figure 17 (Part 1 of 3). 101- and 102-Key Keyboard Scan-Code Set 2

Key #	101-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
47	X	22/F0 22	**	**	**	**
48	C	21/F0 21	**	**	**	**
49	V	2A/F0 2A	**	**	**	**
50	B	32/F0 32	**	**	**	**
51	N	31/F0 31	**	**	**	**
52	M	3A/F0 3A	**	**	**	**
53	,	41/F0 41	**	**	**	**
54	.	49/F0 49	**	**	**	**
55	/	4A/F0 4A	**	**	**	**
57	R. Shift*	59/F0 59	**	**	**	**
58	L. Ctrl*	14/F0 14	**	**	**	**
60	L. Alt*	11/F0 11	**	**	**	**
61	Space Bar	29/F0 29	**	**	**	**
62	R. Alt*	E0 11/ E0 F0 11	**	**	**	**
64	R. Ctrl*	E0 14/ E0 F0 14	**	**	**	**
75	Insert	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	Delete	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	Home	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	End	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	Page Up	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	Page Down	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	**	**

Figure 17 (Part 2 of 3). 101- and 102-Key Keyboard Scan-Code Set 2

Key #	101-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
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	**	**
90	Num Lock*	77/F0 77	**	**	**	**
91	K.P. Home	6C/F0 6C	**	**	**	**
92	K.P. ←	6B/F0 6B	**	**	**	**
93	K.P. End	69/F0 69	**	**	**	**
95	/	E0 4A/ E0 F0 4A	E0 F0 12 E0 4A/ E0 F0 4A E0 12	**	**	**
96	K.P. ↑	75/F0 75	**	**	**	**
97	K.P. 5	73/F0 73	**	**	**	**
98	K.P. ↓	72/F0 72	**	**	**	**
99	K.P. Ins	70/F0 70	**	**	**	**
100	K.P. *	7C/F0 7C	**	**	**	**
101	K.P. Pgup	7D/F0 7D	**	**	**	**
102	K.P. →	74/F0 74	**	**	**	**
103	K.P. Pgdn	7A/F0 7A	**	**	**	**
104	K.P. Del	71/F0 71	**	**	**	**
105	K.P. -	7B/F0 7B	**	**	**	**
106	K.P. +	E0 5A/ E0 F0 5A	**	**	**	**
108	K.P. Enter	E0 5A/ E0 F0 5A	**	**	**	**
110	Esc	76/F0 76	**	**	**	**
112	F1	05/F0 05	**	**	**	**
113	F2	06/F0 06	**	**	**	**
114	F3	04/F0 04	**	**	**	**
115	F4	0C/F0 0C	**	**	**	**
116	F5	03/F0 03	**	**	**	**
117	F6	0B/F0 0B	**	**	**	**
118	F7	83/F0 83	**	**	**	**
119	F8	0A/F0 0A	**	**	**	**
120	F9	01/F0 01	**	**	**	**
121	F10	09/F0 09	**	**	**	**
122	F11	78/F0 78	**	**	**	**
123	F12	07/F0 07	**	**	**	**
124	Print Scr/SysReq	E0 12 E0 7C/E0 F0 7C E0 F0 12	E0 7C/ E0 F0 7C (see note ***)	**	84/F0 84 (see note***)	E0/7C E0 F0 7C (see note ***)
125	Scrl Lock	7E/F0 7E	**	**	**	**
126	Pause/ Break	E1 14 77 E1 F0 14 F0 77	**	**	**	**

* = May cause case change (See Column Heading to determine proper case).
** = Same as base case.
*** = Num Lock does not cancel L. Shift or R. Shift, L. Shift and R. Shift do not change Alt or Ctrl cases.
**** = Key does not exist in 101-key keyboard, but does exist in 102-key keyboard.

Figure 17 (Part 3 of 3). 101- and 102-Key Keyboard Scan-Code Set 2

84- and 85-Key Keyboard Scan-Code Set 2

Key #	84-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
8	7/7	3D/F0 3D	**	6C/F0 6C	**	**
9	8/8	3E/F0 3E	**	75/F0 75	**	**
10	9/9	46/F0 46	**	7D/F0 7D	**	**
12	-/-	4E/F0 4E	**	7B/F0 7B	**	**
13	=/+	55/F0 55	**	79/F0 79	**	**
23	U/4	3C/F0 3C	**	3C/F0 3C	**	**
24	I/5	43/F0 43	**	73/F0 73	**	**
25	O/6	44/F0 44	**	74/F0 74	**	**
37	J/1	3B/F0 3B	**	69/F0 69	**	**
38	K/2	42/F0 42	**	72/F0 72	**	**
39	L/3	4B/F0 4B	**	7A/F0 7A	**	**
40	;/*	4C/F0 4C	**	7C/F0 7C	**	**
43	Enter	5A/F0 5A	**	E0 5A/E0 F0 5A	**	**
52	M/0	3A/F0 3A	**	70/F0 70	**	**
54	./.	49/F0 49	**	71/F0 71	**	**
55	/	4A/F0 4A	E0 F0 12 E0 4A/ E0 F0 4A E0 12 (With Num Lock on)	E0 4A/E0 F0 4A	**	**
125	Scrl Lock	7E/F0 7E	77/F0 77 *	**	**	**

* = Key 125 becomes Num Lock in the shift case
** = Same as base case

Figure 18. 84- and 85-Key Keyboard Scan-Code Set 2. These are exceptions to the 101/102-key keyboard.

122-Key Keyboard Scan-Code Set 2

Key #	122-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
1	'	0E/F0 0E	**	**	**	**
2	1	16/F0 16	**	**	**	**
3	2	1E/F0 1E	**	**	**	**
4	3	26/F0 26	**	**	**	**
5	4	25/F0 25	**	**	**	**
6	5	2E/F0 2E	**	**	**	**
7	6	36/F0 36	**	**	**	**
8	7	3D/F0 3D	**	**	**	**
9	8	3E/F0 3E	**	**	**	**
10	9	46/F0 46	**	**	**	**
11	0	45/F0 45	**	**	**	**
12	-	4E/F0 4E	**	**	**	**
13	=	55/F0 55	**	**	**	**
15	Backspace	66/F0 66	**	**	**	**
16	Tab	0D/F0 0D	**	**	**	**
17	Q	15/F0 15	**	**	**	**
18	W	1D/F0 1D	**	**	**	**
19	E	24/F0 24	**	**	**	**
20	R	2D/F0 2D	**	**	**	**
21	T	2C/F0 2C	**	**	**	**
22	Y	35/F0 35	**	**	**	**
23	U	3C/F0 3C	**	**	**	**
24	I	43/F0 43	**	**	**	**
25	O	44/F0 44	**	**	**	**
26	P	4D/F0 4D	**	**	**	**
27	[	54/F0 54	**	**	**	**
28	]	5B/F0 5B	**	**	**	**
30	Caps Lock	58/F0 58	**	**	**	**
31	A	1C/F0 1C	**	**	**	**
32	S	1B/F0 1B	**	**	**	**
33	D	23/F0 23	**	**	**	**
34	F	2B/F0 2B	**	**	**	**
35	G	34/F0 34	**	**	**	**
36	H	33/F0 33	**	**	**	**
37	J	3B/F0 3B	**	**	**	**
38	K	42/F0 42	**	**	**	**
39	L	4B/F0 4B	**	**	**	**
40	;	4C/F0 4C	**	**	**	**
41	'	52/F0 52	**	**	**	**
42	\	5D/F0 5D	**	**	**	**
43	Enter	5A/F0 5A	**	**	**	**
44	L. Shift*	12/F0 12	**	**	**	**
45	\	61/F0 61	**	**	**	**

Figure 19 (Part 1 of 4). 122-Key Keyboard Scan-Code Set 2

Key #	122-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
46	Z	1A/F0 1A	**	**	**	**
47	X	22/F0 22	**	**	**	**
48	C	21/F0 21	**	**	**	**
49	V	2A/F0 2A	**	**	**	**
50	B	32/F0 32	**	**	**	**
51	N	31/F0 31	**	**	**	**
52	M	3A/F0 3A	**	**	**	**
53	,	41/F0 41	**	**	**	**
54	.	49/F0 49	**	**	**	**
55	/	4A/F0 4A	**	**	**	**
57	R. Shift*	59/F0 59	**	**	**	**
58	L. Ctrl*	14/F0 14	**	**	**	**
60	L. Alt*	11/F0 11	**	**	**	**
61	Space Bar	29/F0 29	**	**	**	**
62	R. Alt*	E0 11/E0 F0 11	**	**	**	**
64	R. Ctrl*	E0 14/E0 F0 14	**	**	**	**
65	Clear	5F/F0 5F	**	**	**	**
66	Pause	E1 14 77 E1 F0 14 F0 77	**	**	**	E1 77 E1 F0 77
67	ErEOF	50/F0 50	**	**	**	**
68	Copy/Play	6F/F0 6F	**	**	**	**
69	(No Key)	48/F0 48	**	**	**	**
70	Attn/Sysr	19/F0 19	**	**	**	**
71	CrSel	39/F0 39	**	**	**	**
72	ExSel	53/F0 53	**	**	**	**
73	PrtSc	E0 12 E0 7C E0 F0 7C E0 F0 12	E0 7C/E0 F0 7C (See note ***)	**	E0 7C/E0 F0 7C(See note ***)	E0 7C/E0 F0 7C (See note ***)
74	(No Key)	5C/F0 5C	**	**	**	**
75	DupPA1	17/F0 17	**	**	**	**
76	End	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	**	**
78	←	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	PgUp	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	**	**

Figure 19 (Part 2 of 4). 122-Key Keyboard Scan-Code Set 2

Key #	122-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
81	Ins	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	**	**
82	↑	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	**	**
83	Home	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	**	**
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	PgDn	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	**	**
86	Del	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	**	**
88	→	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	**	**
90	Esc	76/F0 76	**	**	**	**
91	K.P. Home/7	6C/F0 6C	**	**	**	**
92	K.P. ← /4	6B/F0 6B	**	**	**	**
93	K.P. End/1	69/F0 69	**	**	**	**
95	ScrkLk/NumL	7E/F0 7E	77/F0 77	7E/F0 7E	7E/F0 7E	7E/F0 7E
96	K.P. ↑/8	75/F0 75	**	**	**	**
97	K.P. 5	73/F0 73	**	**	**	**
98	K.P. ↓/2	72/F0 72	**	**	**	**
99	K.P. Ins/0	70/F0 70	**	**	**	**
100	K.P. *	7C/F0 7C	**	**	**	**
101	K.P. PgUp/9	7D/F0 7D	**	**	**	**
102	K.P. →/6	74/F0 74	**	**	**	**
103	K.P. ↓/3	7A/F0 7A	**	**	**	**
104	K.P. Del/.	71/F0 71	**	**	**	**
105	Space/Break	E0 29/E0 F0 29	**	**	**	E0 7E/ E0 F0 7E
106	K.P. +	79/F0 79	**	**	**	**
107	K.P. -	7B/F0 7B	**	**	**	**
108	K.P. Enter	E0 5A/E0 F0 5A	**	**	**	**

Figure 19 (Part 3 of 4). 122-Key Keyboard Scan-Code Set 2

Key #	122-Key U.S. Legend	Base Case	Shift Case (Left)	Num Lock Case	Alt Case	Ctrl Case
110	F1	05/F0 05	**	**	**	**
111	F2	06/F0 06	**	**	**	**
112	F3	04/F0 04	**	**	**	**
113	F4	0C/F0 0C	**	**	**	**
114	F5	03/F0 03	**	**	**	**
115	F6	0B/F0 0B	**	**	**	**
116	F7	83/F0 83	**	**	**	**
117	F8	0A/F0 0A	**	**	**	**
118	F9	01/F0 01	**	**	**	**
119	F10	09/F0 09	**	**	**	**
120	F11	78/F0 78	**	**	**	**
121	F12	07/F0 07	**	**	**	**
122	F13	1F/F0 1F	**	**	**	**
123	F14	27/F0 27	**	**	**	**
124	F15	2F/F0 2F	**	**	**	**
125	F16	5E/F0 5E	**	**	**	**
126	F17	08/F0 08	**	**	**	**
127	F18	10/F0 10	**	**	**	**
128	F19	18/F0 18	**	**	**	**
129	F20	20/F0 20	**	**	**	**
130	F21	28/F0 28	**	**	**	**
131	F22	30/F0 30	**	**	**	**
132	F23	38/F0 38	**	**	**	**
133	F24	40/F0 40	**	**	**	**

* = May cause case change (See column heading to determine proper case)
** = Same as base case
***= Num Lock does not cancel L. Shift or R. Shift, L. Shift and R. Shift do not change Alt or Ctrl cases

Figure 19 (Part 4 of 4). 122-Key Keyboard Scan-Code Set 2

101- and 102-Key Keyboard Scan-Code Set 3

Key #	101-Key U.S. Legend	Base Case	Default Key Type
1	'	0E/F0 0E	Typematic
2	1	16/F0 16	Typematic
3	2	1E/F0 1E	Typematic
4	3	26/F0 26	Typematic
5	4	25/F0 25	Typematic
6	5	2E/F0 2E	Typematic
7	6	36/F0 36	Typematic
8	7	3D/F0 3D	Typematic
9	8	3E/F0 3E	Typematic
10	9	46/F0 46	Typematic
11	0	45/F0 45	Typematic
12	-	4E/F0 4E	Typematic
13	=	55/F0 55	Typematic
15	Backspace	66/F0 66	Typematic
16	Tab	0D/F0 0D	Typematic
17	Q	15/F0 15	Typematic
18	W	1D/F0 1D	Typematic
19	E	24/F0 24	Typematic
20	R	2D/F0 2D	Typematic
21	T	2C/F0 2C	Typematic
22	Y	35/F0 35	Typematic
23	U	3C/F0 3C	Typematic
24	I	43/F0 43	Typematic
25	O	44/F0 44	Typematic
26	P	4D/F0 4D	Typematic
27	[	54/F0 54	Typematic
28	]	5B/F0 5B	Typematic
29	\	5C/F0 5C	Typematic
30	Caps Lock	14/F0 14	Make/Break
31	A	1C/F0 1C	Typematic
32	S	1B/F0 1B	Typematic
33	D	23/F0 23	Typematic
34	F	2B/F0 2B	Typematic
35	G	34/F0 34	Typematic
36	H	33/F0 33	Typematic
37	J	3B/F0 3B	Typematic
38	K	42/F0 42	Typematic
39	L	4B/F0 4B	Typematic
40	;	4C/F0 4C	Typematic

Figure 20 (Part 1 of 3). 101- and 102-Key Keyboard Scan-Code Set 3

Key #	101-Key U.S. Legend	Base Case	Default Key Type
41	'	52/F0 52	Typematic
42	(No Key)	53/F0 53	Typematic
43	Enter	5A/F0 5A	Typematic
44	L. Shift	12/F0 12	Make/Break
45	(No Key)	13/F0 13	Typematic
46	Z	1A/F0 1A	Typematic
47	X	22/F0 22	Typematic
48	C	21/F0 21	Typematic
49	V	2A/F0 2A	Typematic
50	B	32/F0 32	Typematic
51	N	31/F0 31	Typematic
52	M	3A/F0 3A	Typematic
53	,	41/F0 41	Typematic
54	.	49/F0 49	Typematic
55	/	4A/F0 4A	Typematic
57	R. Shift	59/F0 59	Make/Break
58	L. Ctrl	11/F0 11	Make/Break
60	L. Alt	19/F0 19	Make/Break
61	Space Bar	29/F0 29	Typematic
62	R. Alt	39/F0 39	Make only
64	R. Ctrl	58/F0 58	Make only
75	Insert	67/F0 67	Make only
76	Delete	64/F0 64	Typematic
79	◄	61/F0 61	Typematic
80	Home	6E/F0 6E	Make only
81	End	65/F0 65	Make only
83	↑	63/F0 63	Typematic
84	↓	60/F0 60	Typematic
85	Page Up	6F/F0 6F	Make only
86	Page Down	6D/F0 6D	Make only
89	►	6A/F0 6A	Typematic
90	Num Lock	76/F0 76	Make only
91	K.P. Home	6C/F0 6C	Make only
92	K.P. ◄	6B/F0 6B	Make only
93	K.P. End	69/F0 69	Make only
95	/	77/F0 77	Make only
96	K.P. ↑	75/F0 75	Make only
97	K.P. 5	73/F0 73	Make only
98	K.P. ↓	72/F0 72	Make only
99	K.P. Ins	70/F0 70	Make only
100	K.P. *	7E/F0 7E	Make only

Figure 20 (Part 2 of 3). 101- and 102-Key Keyboard Scan-Code Set 3

Key #	101-Key U.S. Legend	Base Case	Default Key Type
101	K.P. Pgup	7D/F0 7D	Make only
102	K.P. ►	74/F0 74	Make only
103	K.P. Pgdn	7A/F0 7A	Make only
104	K.P. Del	71/F0 71	Make only
105	K.P. -	84/F0 84	Make only
106	K.P. +	7C/F0 7C	Typematic
108	K.P. Enter	79/F0 79	Make only
110	Esc	08/F0 08	Make only
112	F1	07/F0 07	Make only
113	F2	0F/F0 0F	Make only
114	F3	17/F0 17	Make only
115	F4	1F/F0 1F	Make only
116	F5	27/F0 27	Make only
117	F6	2F/F0 2F	Make only
118	F7	37/F0 37	Make only
119	F8	3F/F0 3F	Make only
120	F9	47/F0 47	Make only
121	F10	4F/F0 4F	Make only
122	F11	56/F0 56	Make only
123	F12	5E/F0 5E	Make only
124	Prt Screen	57 F0 57	Make only
125	Scrl Lock	5F/F0 5F	Make only
126	Pause	62/F0 62	Make only

Figure 20 (Part 3 of 3). 101- and 102-Key Keyboard Scan-Code Set 3

122-Key Keyboard Scan-Code Set 3

Key #	122-Key U.S. Legend	Base Case	Default Key Type
1	'	0E/F0 0E	Typematic
2	1	16/F0 16	Typematic
3	2	1E/F0 1E	Typematic
4	3	26/F0 26	Typematic
5	4	25/F0 25	Typematic
6	5	2E/F0 2E	Typematic
7	6	36/F0 36	Typematic
8	7	3D/F0 3D	Typematic
9	8	3E/F0 3E	Typematic
10	9	46/F0 46	Typematic
11	0	45/F0 45	Typematic
12	-	4E/F0 4E	Typematic
13	=	55/F0 55	Typematic
14	(No Key)	5D/F0 5D	
15	Backspace	66/F0 66	Typematic
16	Tab	0D/F0 0D	Typematic
17	Q	15/F0 15	Typematic
18	W	1D/F0 1D	Typematic
19	E	24/F0 24	Typematic
20	R	2D/F0 2D	Typematic
21	T	2C/F0 2C	Typematic
22	Y	35/F0 35	Typematic
23	U	3C/F0 3C	Typematic
24	I	43/F0 43	Typematic
25	O	44/F0 44	Typematic
26	P	4D/F0 4D	Typematic
27	[	54/F0 54	Typematic
28	]	5B/F0 5B	Typematic
29	(No Key)	5C/F0 5C	Typematic
30	Caps Lock	14/F0 14	Make/Break
31	A	1C/F0 1C	Typematic
32	S	1B/F0 1B	Typematic
33	D	23/F0 23	Typematic
34	F	2B/F0 2B	Typematic
35	G	34/F0 34	Typematic
36	H	33/F0 33	Typematic
37	J	3B/F0 3B	Typematic
38	K	42/F0 42	Typematic
39	L	4B/F0 4B	Typematic
40	;	4C/F0 4C	Typematic

Figure 21 (Part 1 of 3). 122-Key Keyboard Scan-Code Set 3

Key #	122-Key U.S. Legend	Base Case	Default Key Type
41	'	52/F0 52	Typematic
42	\	53/F0 53	Typematic
43	Enter	5A/F0 5A	Typematic
44	L. Shift	12/F0 12	Make/Break
45	(No Key)	13/F0 13	Typematic
46	Z	1A/F0 1A	Typematic
47	X	22/F0 22	Typematic
48	C	21/F0 21	Typematic
49	V	2A/F0 2A	Typematic
50	B	32/F0 32	Typematic
51	N	31/F0 31	Typematic
52	M	3A/F0 3A	Typematic
53	,	41/F0 41	Typematic
54	.	49/F0 49	Typematic
55	/	4A/F0 4A	Typematic
56	(No Key)	51/F0 51	
57	R. Shift	59/F0 59	Make/Break
58	L. Ctrl	11/F0 11	Make/Break
60	L. Alt	19/F0 19	Make/Break
61	Space Bar	29/F0 29	Typematic
62	R. Alt	39/F0 39	Make/Break
64	R. Ctrl	58/F0 58	Make/Break
65	(No Key)	06/F0 F6	
66	Pause	0C/F0 0C	Make only
67	(No Key)	0B/F0 0B	
68	(No Key)	0A/F0 0A	
69	(No Key)	09/F0 09	
70	Attn/Sysreq	05/F0 05	Make only
71	(No Key)	04/F0 04	
72	(No Key)	03/F0 03	
73	PrtSc	83/F0 83	Make only
74	(No Key)	01/F0 01	
75	(No Key)	67/F0 67	
76	End	64/F0 64	Typematic
78	←	61/F0 61	Typematic
80	PgUp	6E/F0 6E	Make only
81	Ins	65/F0 65	Make only
82	↑	63/F0 63	Typematic
83	Home	62/F0 62	Make only
84	↓	60/F0 60	Typematic
85	PgDn	6F/F0 6F	Make only
86	Del	6D/F0 6D	Typematic
88	→	6A/F0 6A	Typematic
90	Esc	76/F0 76	Make only

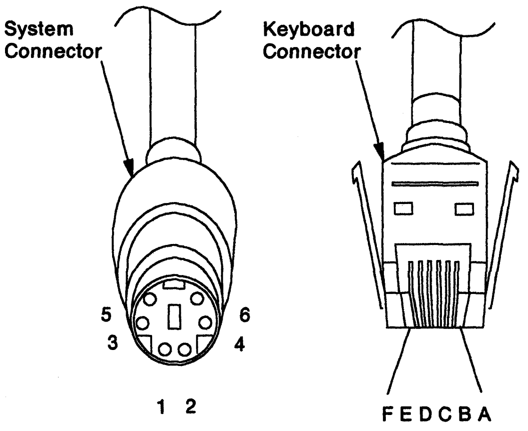
Figure 21 (Part 2 of 3). 122-Key Keyboard Scan-Code Set 3

Key #	122-Key U.S. Legend	Base Case	Default Key Type
91	K.P. Home/7	6C/F0 6C	Make only
92	K.P. ←/4	6B/F0 6B	Make only
93	K.P. End/1	69/F0 69	Make only
94	(No Key)	68/F0 68	
95	ScrLk/NumL	77/F0 77	Make only
96	K.P. ↑/8	75/F0 75	Make only
97	K.P. 5	73/F0 73	Make only
98	K.P. ↓/2	72/F0 72	Make only
99	K.P. Ins/0	70/F0 70	Make only
100	K.P. *	7E/F0 7E	Make only
101	K.P. PgUp/9	7D/F0 7D	Make only
102	K.P. →/6	74/F0 74	Make only
103	K.P. PgDn/3	7A/F0 7A	Make only
104	K.P. Del/.	71/F0 71	Make only
105	Space/Break	84/F0 84	Make only
106	K.P. +	7C/F0 7C	Typematic
107	K.P. -	7B/F0 7B	
108	K.P. Enter	79/F0 79	Make only
109	(No Key)	78/F0 78	
110	F1	07/F0 07	Make only
111	F2	0F/F0 0F	Make only
112	F3	17/F0 17	Make only
113	F4	1F/F0 1F	Make only
114	F5	27/F0 27	Make only
115	F6	2F/F0 2F	Make only
116	F7	37/F0 37	Make only
117	F8	3F/F0 3F	Make only
118	F9	47/F0 47	Make only
119	F10	4F/F0 4F	Make only
120	F11	56/F0 56	Make only
121	F12	5E/F0 5E	Make only
122	F13	08/F0 08	Make only
123	F14	10/F0 10	Make only
124	F15	18/F0 18	Make only
125	F16	20/F0 20	Make only
126	F17	28/F0 28	Make only
127	F18	30/F0 30	Make only
128	F19	38/F0 38	Make only
129	F20	40/F0 40	Make only
130	F21	48/F0 48	Make only
131	F22	50/F0 50	Make only
132	F23	57/F0 57	Make only
133	F24	5F/F0 5F	Make only

Figure 21 (Part 3 of 3). 122-Key Keyboard Scan-Code Set 3

Cables and Connectors

The keyboard cable connects to the system with a 6-pin miniature DIN connector and to the keyboard with a 6-position connector. The following figures show the pin configuration and signal assignments.



DIN Connector Pins	Signal Name	Keyboard Connector Pins
1	+KBD DATA	B
2	Reserved	F
3	Ground	C
4	+5.0 Vdc	E
5	+KBD CLK	D
6	Reserved	A
Shield	Frame Ground	Shield

Figure 22. Keyboard Connectors Signal and Voltage Assignments

Specifications (84/85-Key Keyboard)

Specifications for the keyboard are as follows:

Power Requirements

- +5 V dc ± 10%
- 275 mA.

Size

- Length: 406 mm (16.0 in.)
- Depth: 190 mm (7.5 in.)
- Height: 58 mm (2.3 in.), legs extended.

Weight

- 1.90 kg (4.2 lb).

Specifications (101/102-Key Keyboard)

Specifications for the keyboard are as follows:

Power Requirements

- +5 V dc ± 10%
- 275 mA.

Size

- Length: 492 mm (19.4 in.)
- Depth: 210 mm (8.3 in.)
- Height: 58 mm (2.3 in.), legs extended.

Weight

- 2.25 kg (5.0 lb).

Specifications (122-Key Keyboard)

Specifications for the keyboard are as follows:

Power Requirements

- +5 V dc ± 10%
- 275 mA.

Size

- Length: 532 mm (20.9 in.)
- Depth: 215 mm (8.46 in.)
- Height: 70 mm (2.76 in.), legs extended.

Weight

- 2.32 kg (5.12 lb).

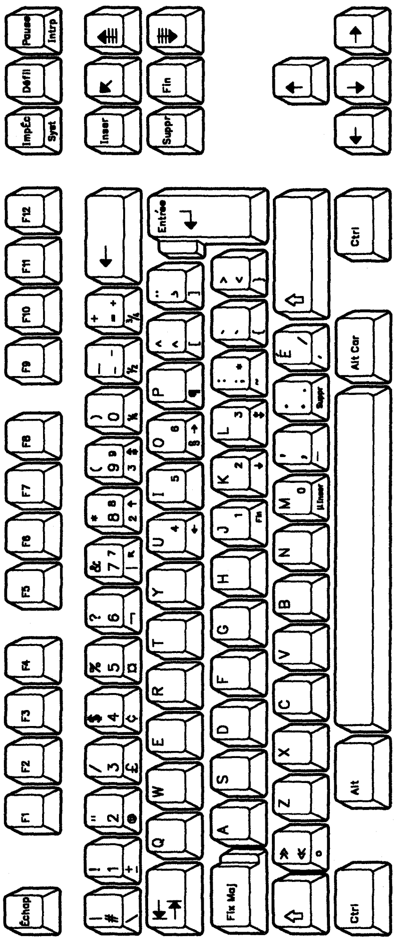
Appendix A. Keyboard Layouts

The keyboard layouts shown on the following pages are presented in alphabetical order, as follows. The number in parenthesis is the language ID number of the country.

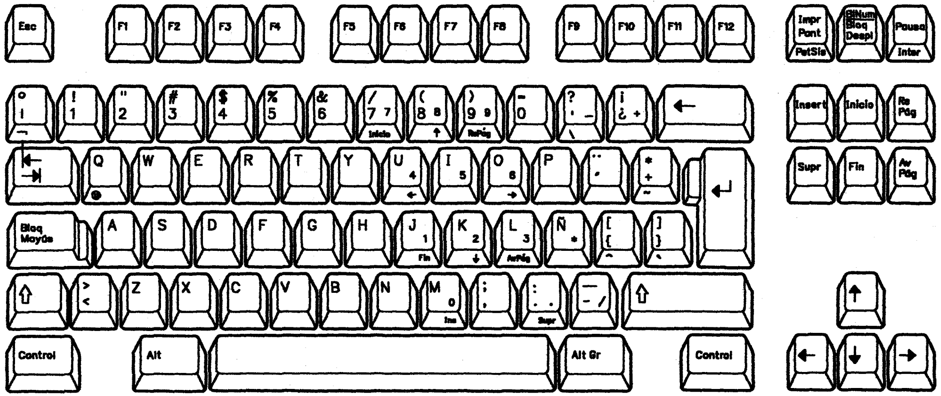
- 84/85-key keyboard layouts:
  - Canadian French
  - Latin America
- 101/102-key keyboard layouts:
  - Arabic (238)
  - Belgium (120)
  - Canada (French) (058)
  - Denmark (159)
  - Finland (153)
  - France (120)
  - France (189)
  - Germany (129)
  - Hebrew (212)
  - Iceland (197)
  - Italy (142)
  - Latin America (171)
  - Netherlands (143)
  - Norway (155)
  - Portugal (163)
  - Spain (172)
  - Switzerland (French) (150F)
  - Switzerland (German) (150G)
  - Turkey (179)
  - United Kingdom (166)
  - Yugoslavia (Latin) (234)
- 122-key keyboard layouts:
  - Belgium (120)
  - Canada (French) (277/058)
  - Denmark (281/159)
  - Finland (285/153)
  - France (251/189)
  - France (251/120)
  - Germany (129)
  - Iceland (197)
  - Italy (293/142)
  - Latin America (309/171)
  - Netherlands (101/143)
  - Norway (281N/155)
  - Portugal (163)

- Spain (071/172)
- Switzerland (French) (150F)
- Switzerland (German) (150G)
- Turkey (402/179)
- United Kingdom (166)
- Yugoslavia (Latin) (234).

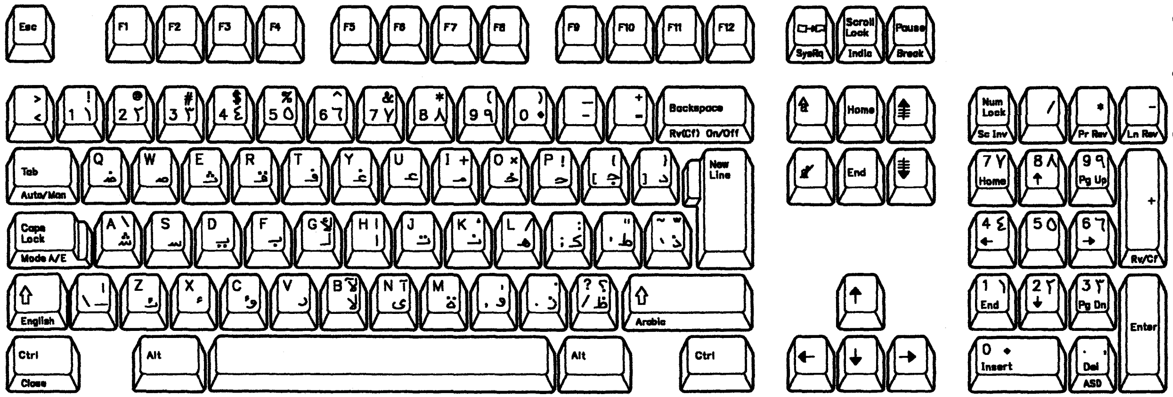
85-Key Canadian French Keyboard



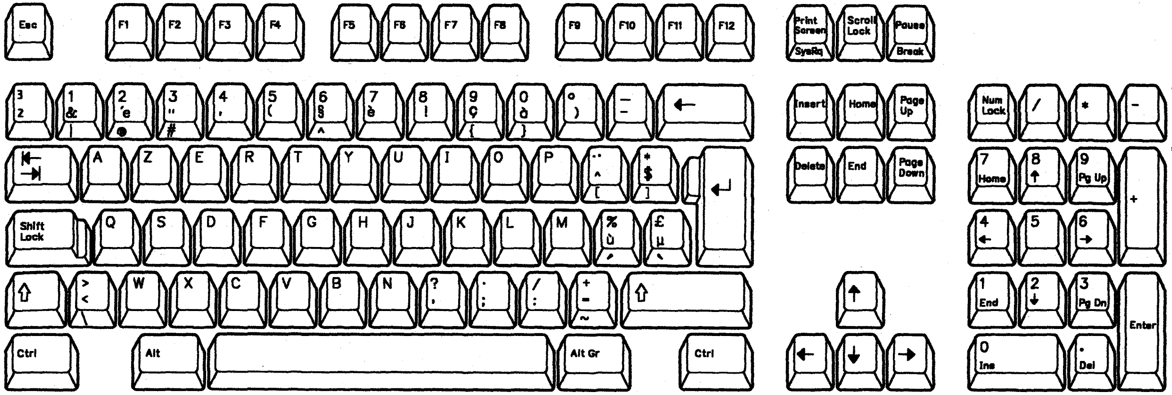
85-Key Latin America Keyboard



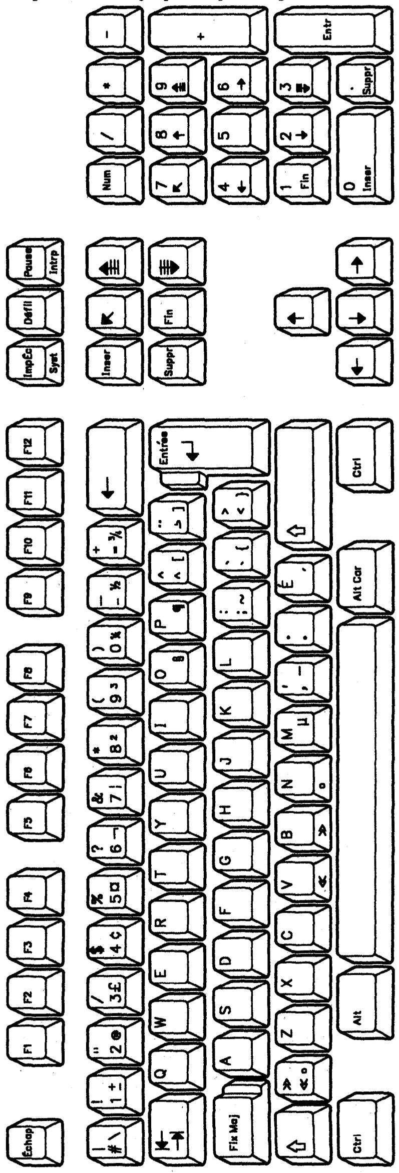
102-Key Arabic (238) Keyboard



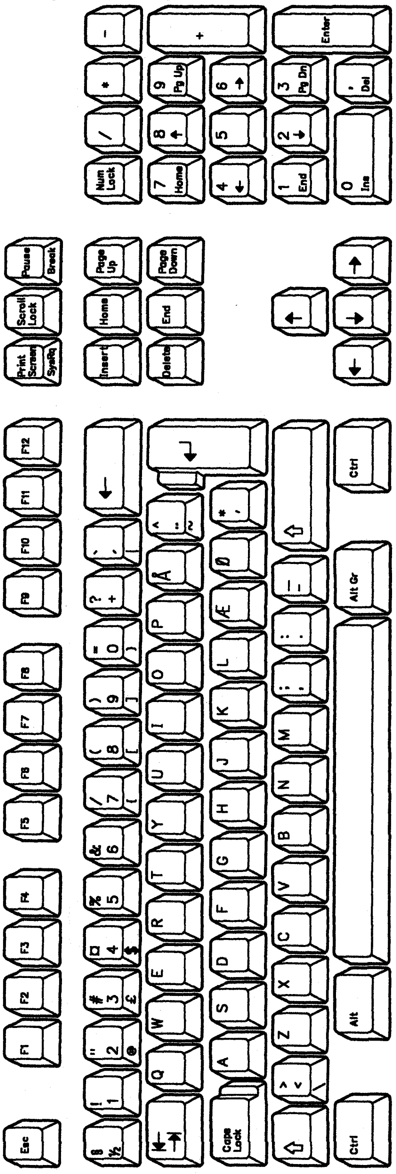
102-Key Belgian (120) Keyboard



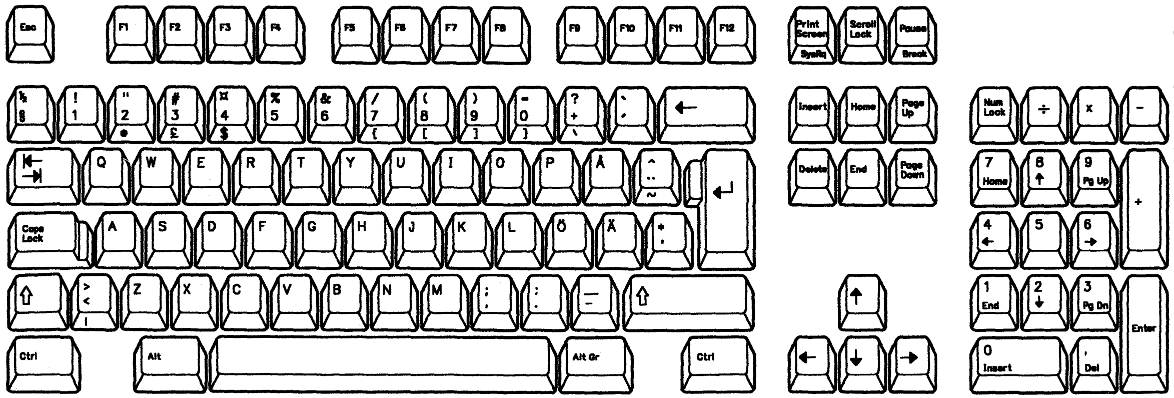
102-Key Canada (French) (058) Keyboard



102-Key Denmark (159) Keyboard

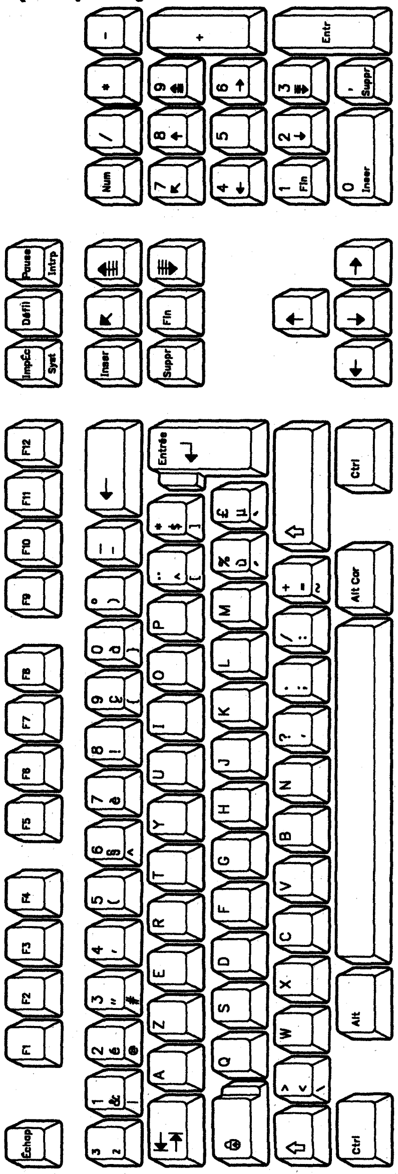


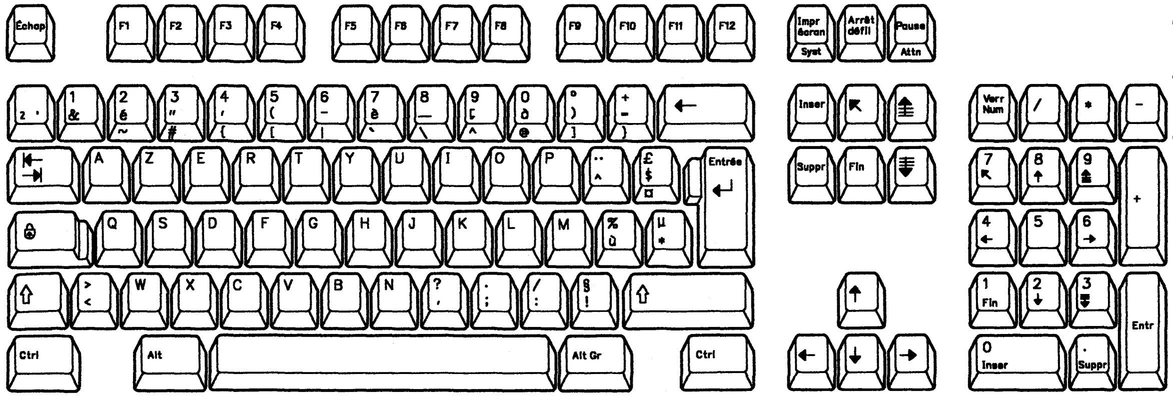
102-Key Finland (153) Keyboard



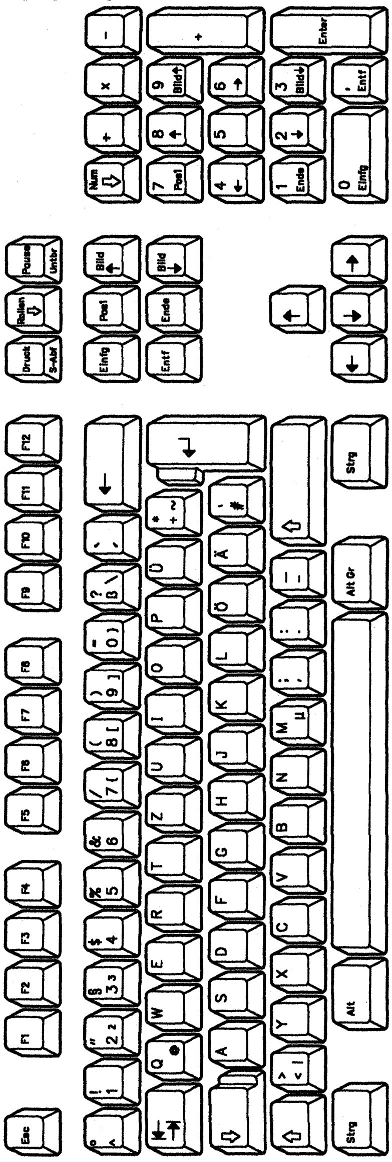
Esc F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Print Screen SysRq | Scroll Lock | Pause Break
½ § | ! 1 | " 2 @ | # 3 £ | ¤ 4 $ | % 5 | & 6 | / 7 { | (8 [|) 9] | = 0 } | ? + \ | ` ´ | ←
Insert | Home | Page Up | Num Lock | ÷ | x | −
Q W E R T Y U I O P Å | ^ ¨ ~ | ↵
Delete | End | Page Down | 7 Home | 8 ↑ | 9 Pg Up | +
Caps Lock | A S D F G H J K L Ö Ä | * '
4 ← | 5 | 6 →
⇧ | > < | | Z X C V B N M | ; , | : . | _ - | ⇧
↑ | 1 End | 2 ↓ | 3 Pg Dn | Enter
Ctrl | Alt | Alt Gr | Ctrl
← ↓ → | 0 Insert | , Del

102-Key France (120) Keyboard



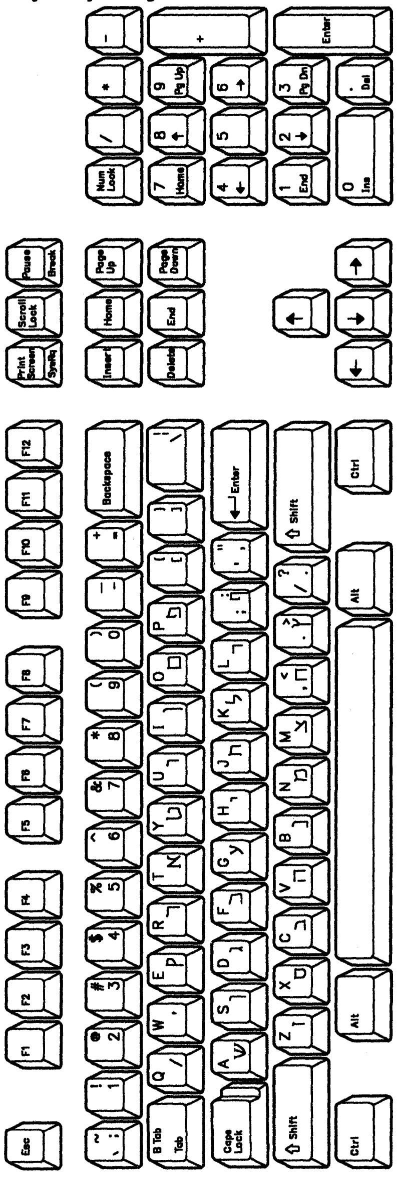

Échap
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Impr écran Syst
Arrêt défil
Pause Attn
² 1 & 2 é ~ 3 " # 4 ' { 5 ([6 - | 7 è ` 8 _ \ 9 ç ^ 0 à @ °)] + = }
Inser
Verr Num
/ * -
A Z E R T Y U I O P ¨ ^ £ $ ¤
Entrée
Suppr
Fin
7 8 9 +
Q S D F G H J K L M % ù µ *
4 5 6
> < W X C V B N ? , . ; / : § !
1 Fin 2 3
Entr
Ctrl Alt Alt Gr Ctrl
0 Inser
. Suppr

102-Key Germany (129) Keyboard



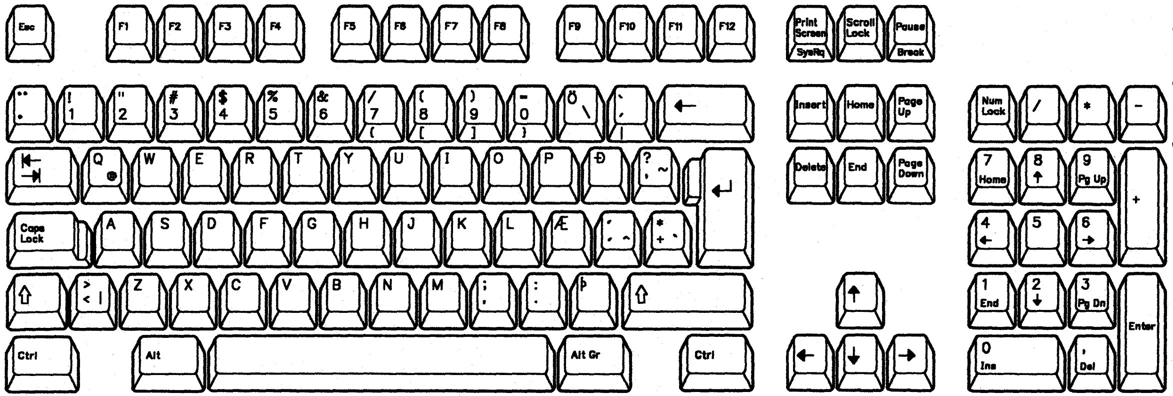
Esc F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Druck S-Abf Rollen Pause Untbr
° ^ ! 1 " 2 ² § 3 ³ $ 4 % 5 & 6 / 7 { (8 [ ) 9] = 0 } ? ß \ ` ´
Q @ W E R T Z U I O P Ü * + ~
A S D F G H J K L Ö Ä ' #
> < | Y X C V B N M µ ; , : . _ -
Strg Alt Alt Gr Strg
Einfg Pos1 Bild Entf Ende Bild
Num x - + 7 Pos1 8 9 Bild 4 5 6 1 Ende 2 3 Bild Enter 0 Einfg , Entf

102-Key Hebrew (212) Keyboard

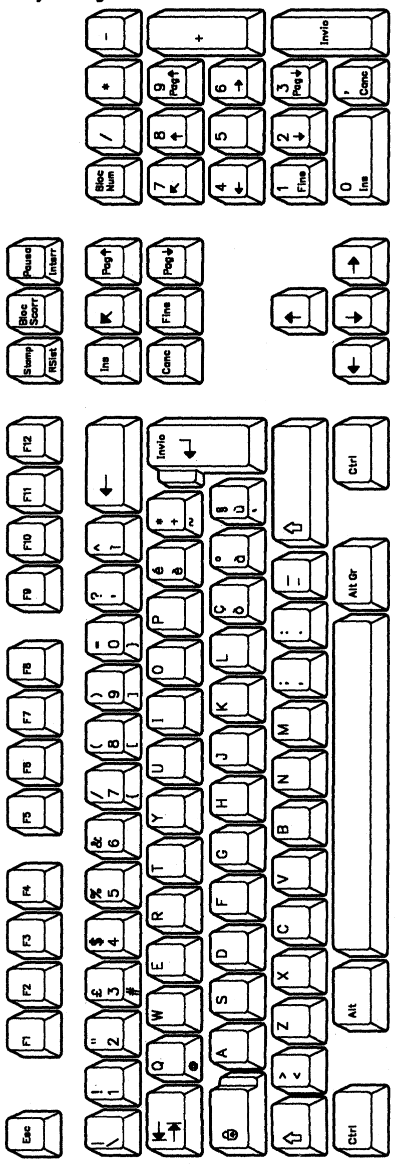


Esc F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Print Screen SysRq, Scroll Lock, Pause Break
~ ; ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Backspace
B Tab Tab, Q /, W ', E ק, R ר, T א, Y ט, U ו, I ן, O ם, P פ, { [, }], | \
Caps Lock, A ש, S ד, D ג, F כ, G ע, H י, J ח, K ל, L ך, : ף, " ,, Enter
Shift, Z ז, X ס, C ב, V ה, B נ, N מ, M צ, < ת, > ץ, ? ., Shift
Ctrl, Alt, Alt, Ctrl
Insert, Home, Page Up, Delete, End, Page Down
Num Lock, /, *, -, 7 Home, 8, 9 Pg Up, +, 4, 5, 6, 1 End, 2, 3 Pg Dn, Enter, 0 Ins, . Del

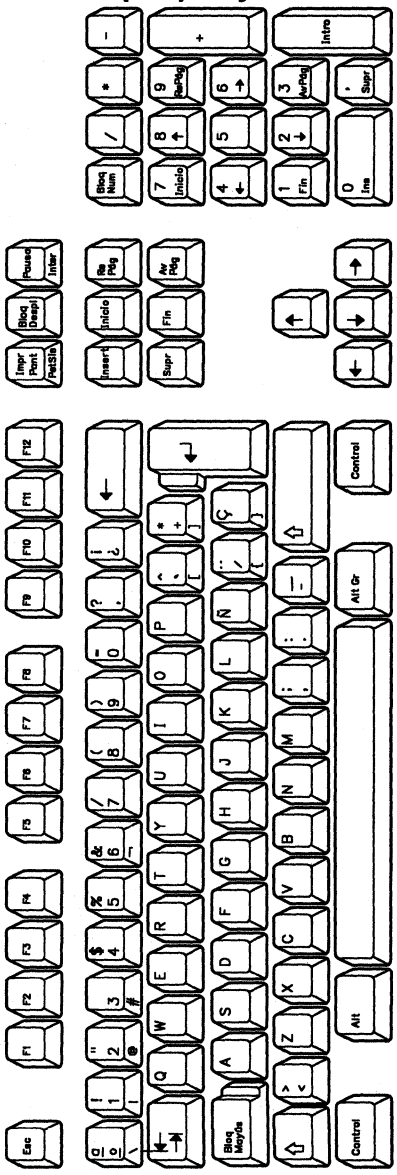
102-Key Iceland (197) Keyboard



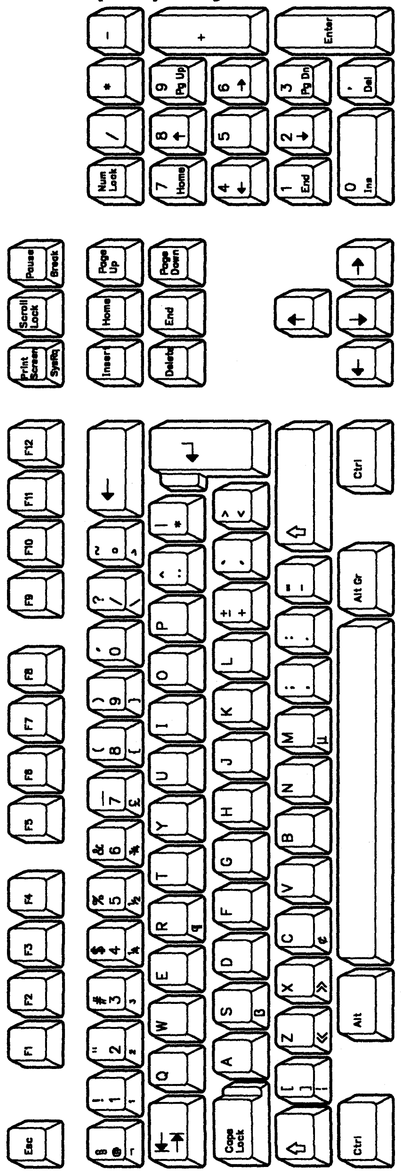
102-Key Italy (142) Keyboard



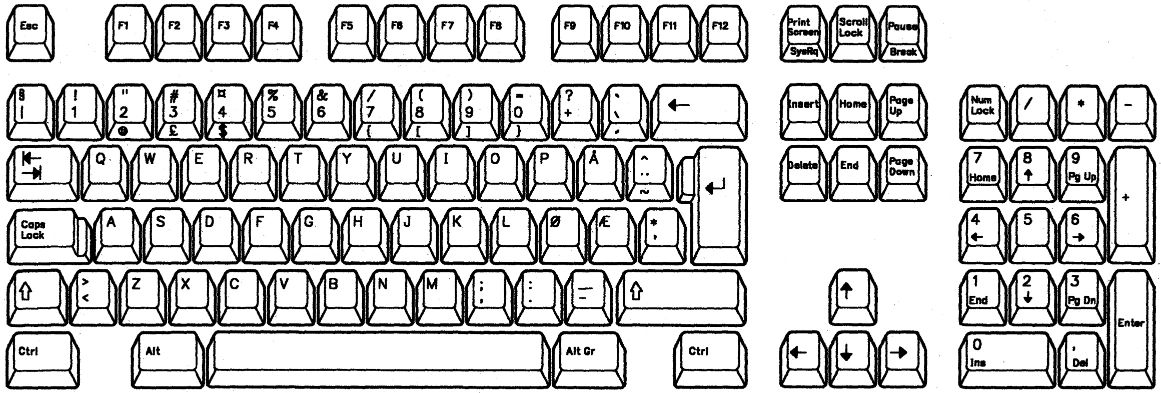
| **102-Key Latin America (171) Keyboard**



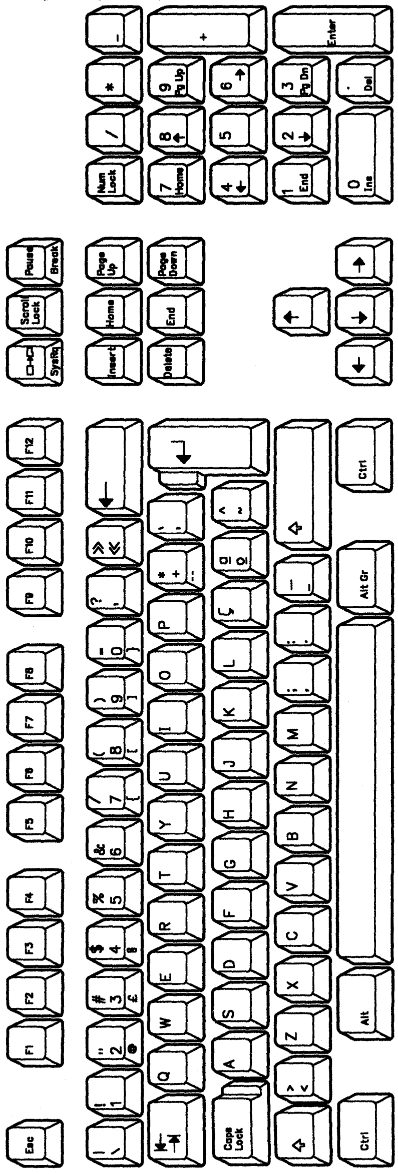
102-Key Netherlands (143) Keyboard



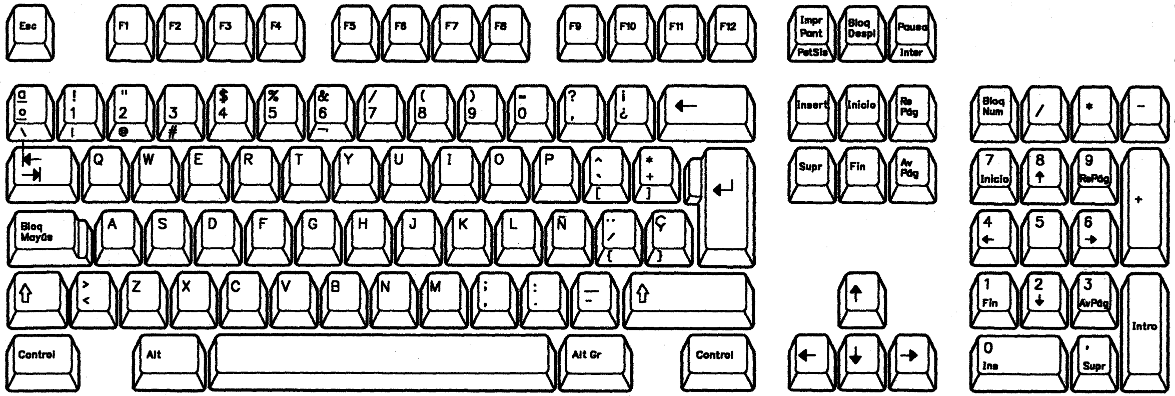
102-Key Norway (155) Keyboard



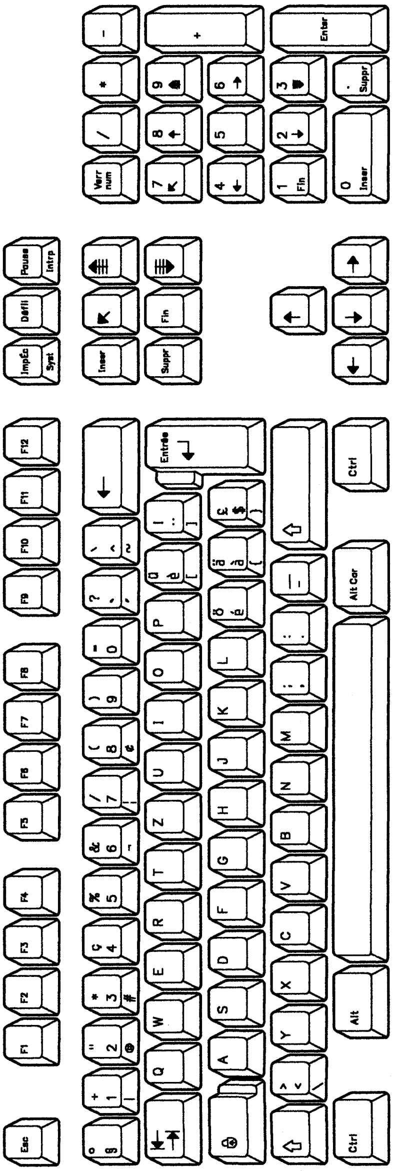
102-Key Portugal (163) Keyboard



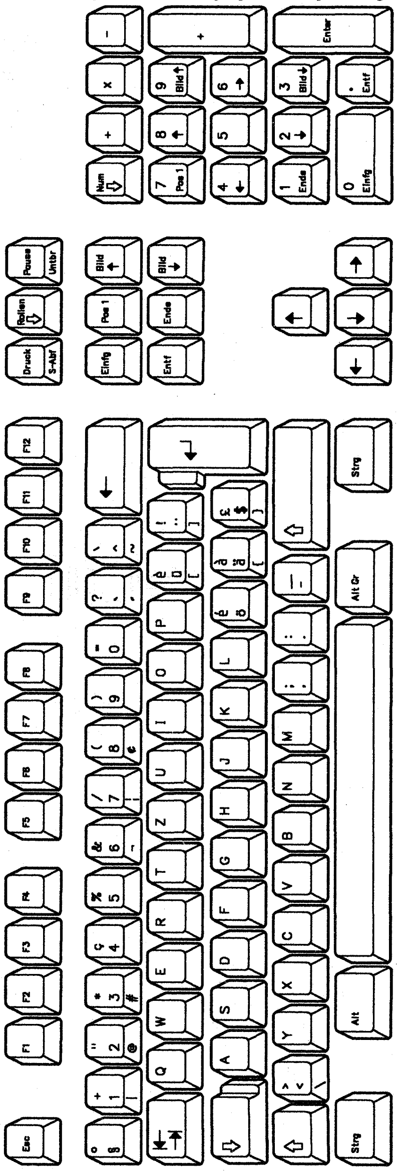
102-Key Spain (172) Keyboard



102-Key Switzerland (French) (150F) Keyboard

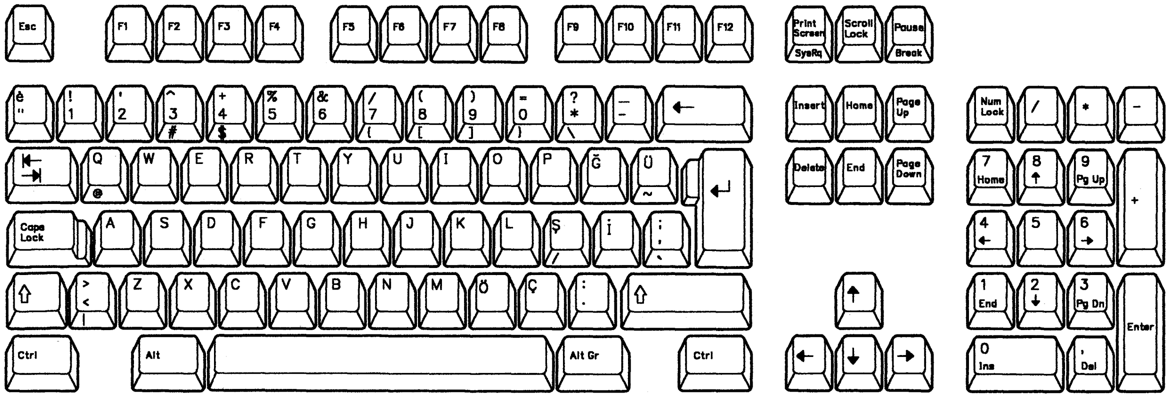


102-Key Switzerland (German) (150G) Keyboard

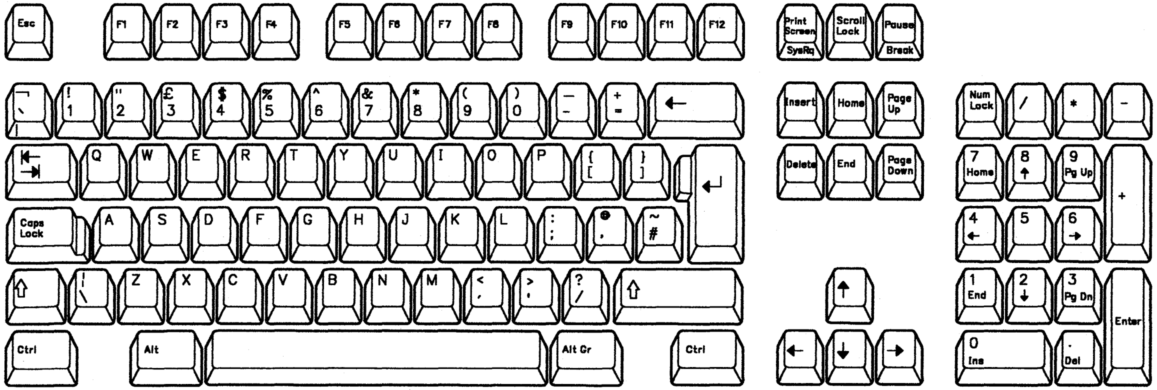


Esc F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Druck S-Abf Rollen Pause Untbr
° § + 1 | " 2 @ * 3 # ç 4 % 5 & 6 ¬ / 7 ¦ (8 ¢) 9 = 0 ? ' ´ ` ^ ~
Q W E R T Z U I O P è ü [! ¨]
A S D F G H J K L é ö à ä { £ $ }
> < \ Y X C V B N M ; , : . _ -
Strg Alt Alt Gr Strg
Einfg Pos 1 Bild ↑ Entf Ende Bild ↓
Num + x -
7 Pos 1 8 9 Bild 4 5 6
1 Ende 2 3 Bild 0 Einfg . Entf Enter +

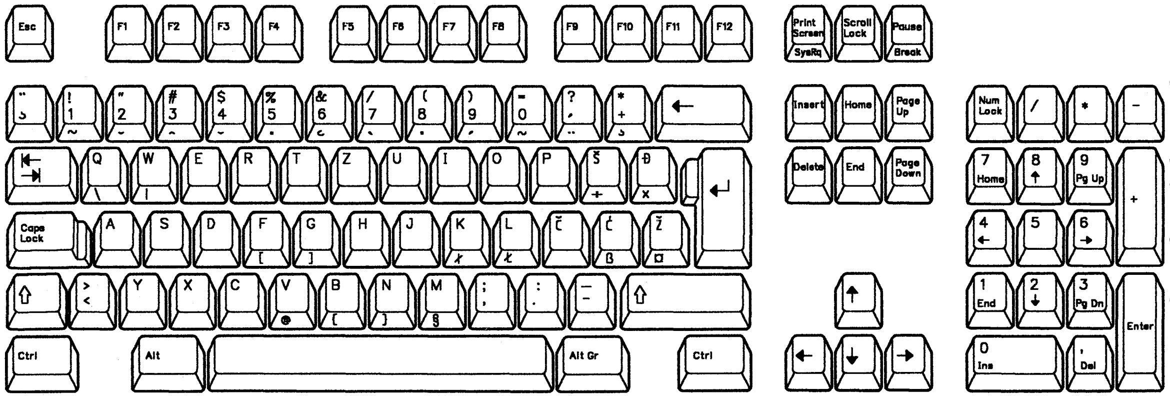
102-Key Turkey (179) Keyboard



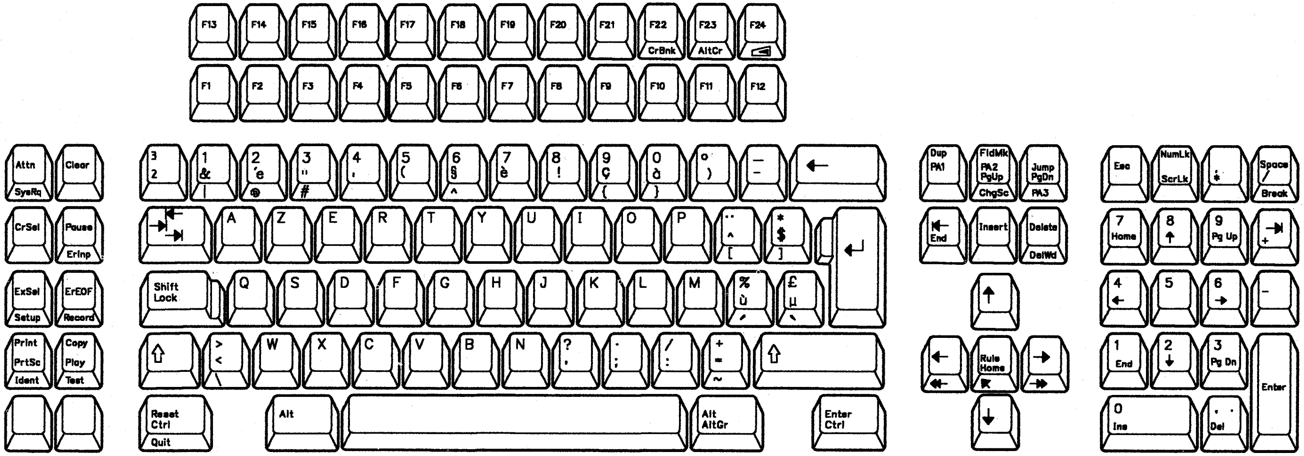
102-Key United Kingdom (166) Keyboard



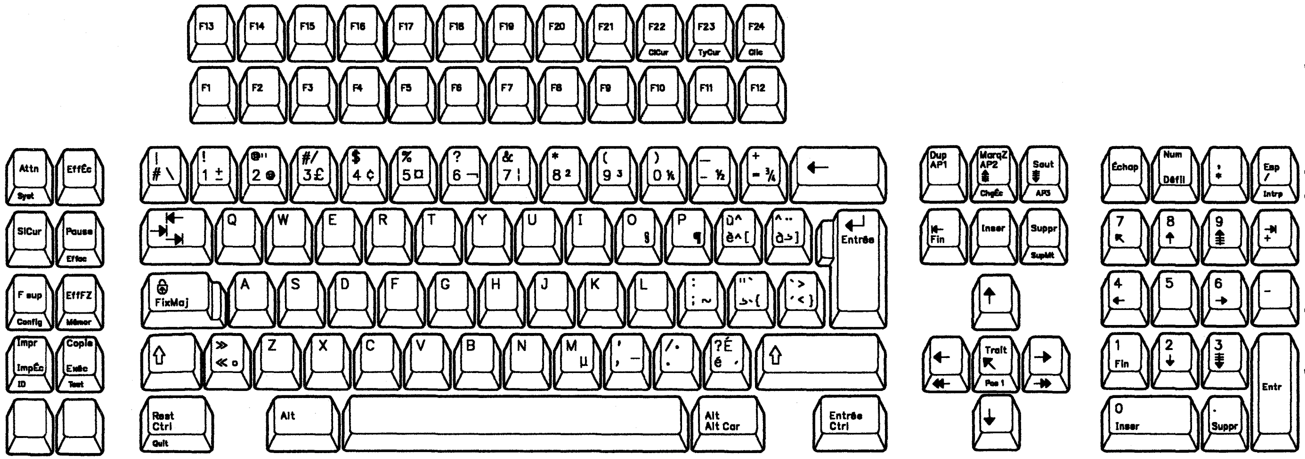
102-Key Yugoslavia (Latin) (234) Keyboard



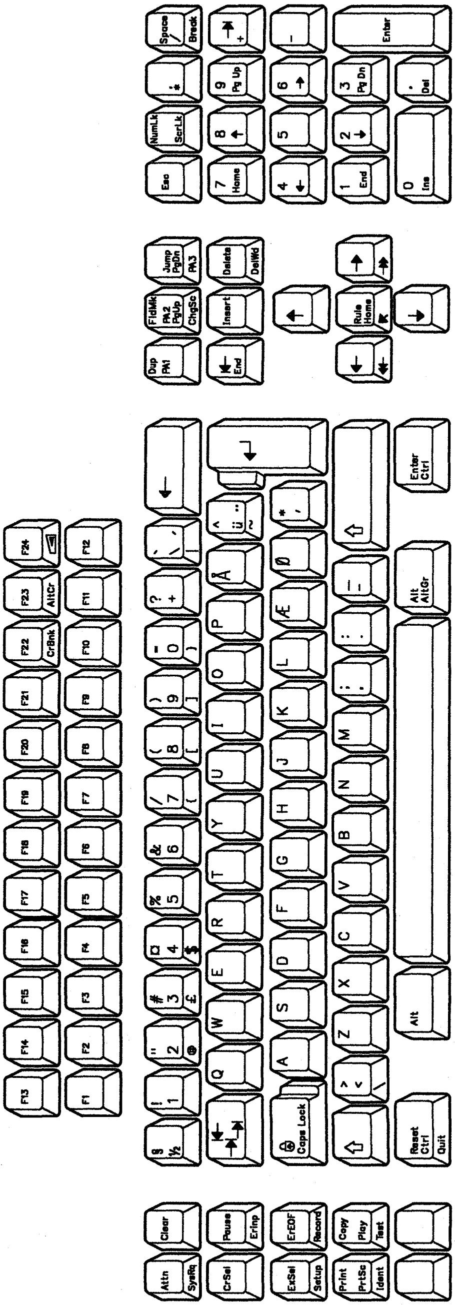
122-Key Belgian (120) Keyboard


F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 CrBnk F23 AltCr F24
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Attn SysRq Clear
CrSel Pause ErInp
ExSel Setup ErEOF Record
Print PrtSc Ident Copy Play Test
³ ² 1 & | 2 é @ 3 " # 4 ' 5 (6 § ^ 7 è 8 ! 9 ç { 0 à } °) _ -
A Z E R T Y U I O P ¨ ^ [* $]
Shift Lock Q S D F G H J K L M % ù ´ £ µ `
> &lt; \ W X C V B N ? , . ; / : + = ~
Reset Ctrl Quit Alt Alt AltGr Enter Ctrl
Dup PA1 FldMk PA2 PgUp ChgSc Jump PgDn PA3
End Insert Delete DelWd
Rule Home
Esc NumLk ScrLk Space / Break
7 Home 8 9 Pg Up +
4 5 6 -
1 End 2 3 Pg Dn Enter
0 Ins . Del

122-Key Canada (French) (277/058) Keyboard

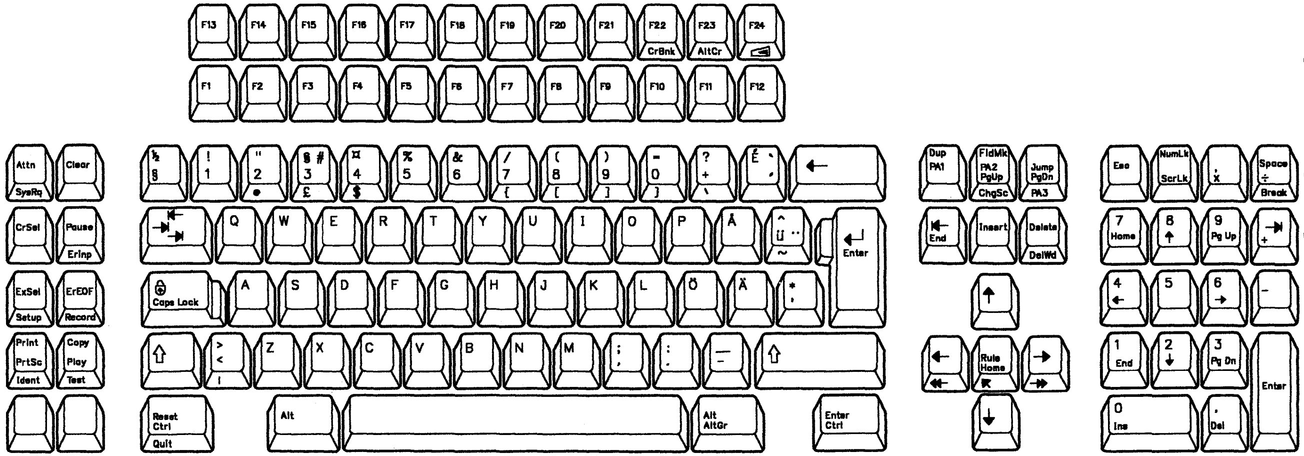


122-Key Denmark (281/159) Keyboard



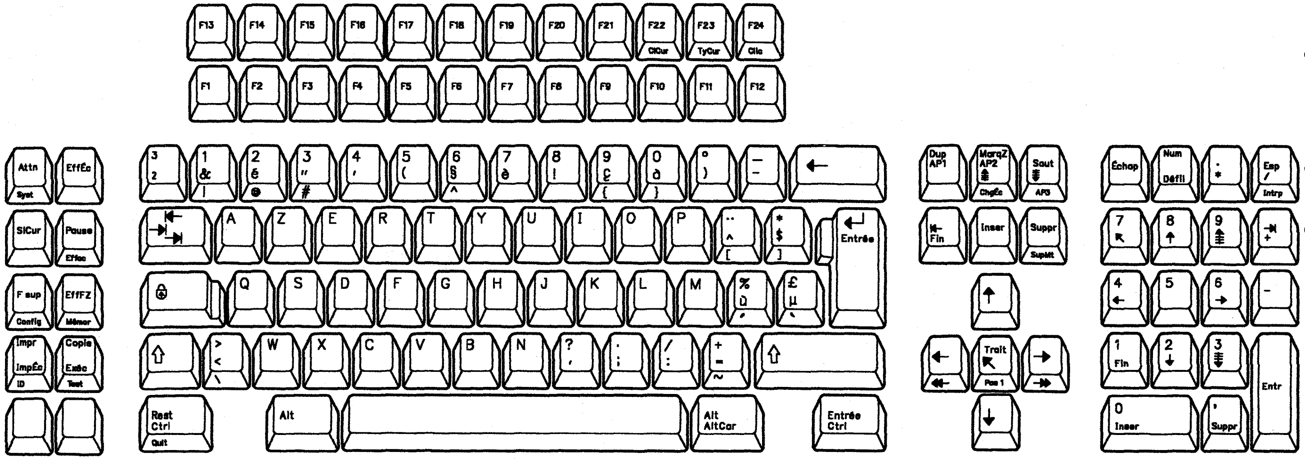
F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 CrBnk F23 AltCr F24
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Attn SysRq Clear § ½ ! 1 " 2 @ # 3 £ ¤ 4 $ % 5 & 6 / 7 { (8 [ ) 9] = 0 } ? + ` \ ´ |
Dup PA1 FldMk PA2 PgUp ChgSc Jump PgDn PA3
Esc NumLk ScrLk * Space / Break
CrSel Pause Erinp Q W E R T Y U I O P Å ^ ü ¨ ~
End Insert Delete DelWd
7 Home 8 9 Pg Up +
ExSel Setup ErEOF Record Caps Lock A S D F G H J K L Æ Ø * '
4 5 6 -
Print PrtSc Ident Copy Play Test > < \ Z X C V B N M ; , : . _ -
Rule Home
1 End 2 3 Pg Dn Enter
Reset Ctrl Quit Alt Alt AltGr Enter Ctrl
0 Ins . Del

122-Key Finland (285/153) Keyboard

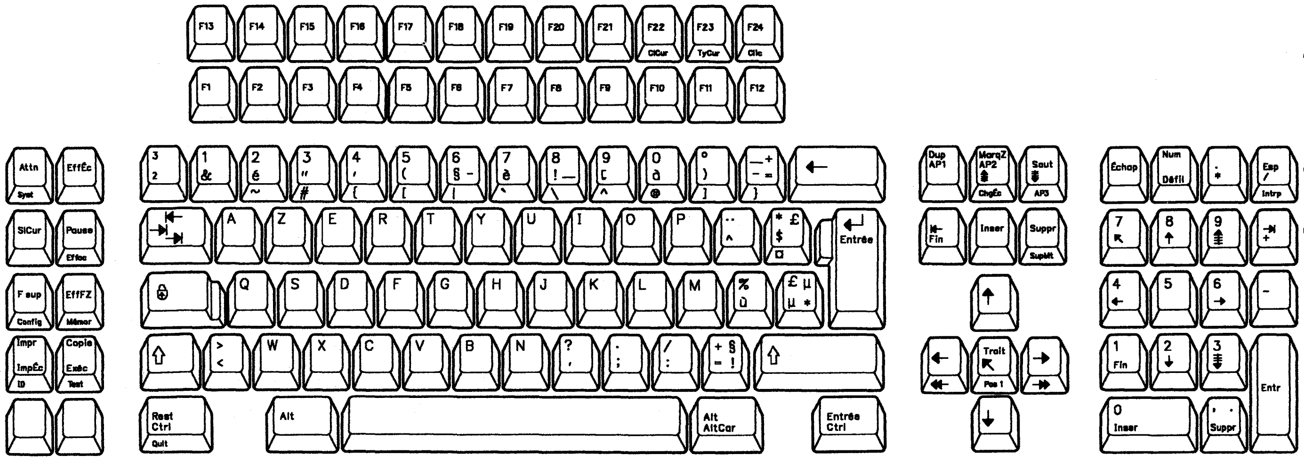


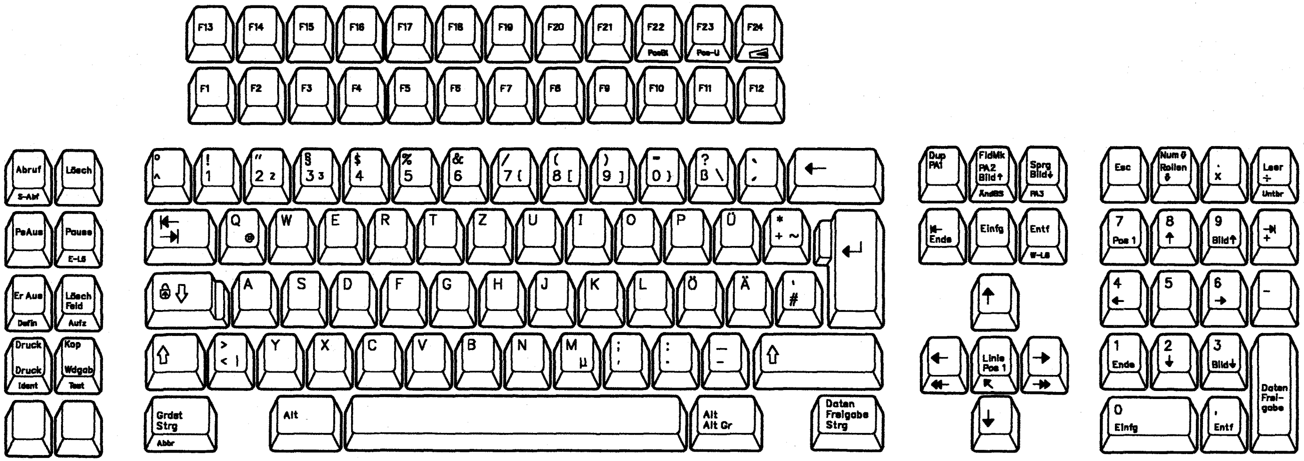
F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 CrBnk F23 AltCr F24
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Attn SysRq Clear
CrSel Pause Erinp
ExSel Setup ErEOF Record
Print PrtSc Ident Copy Play Test
½ § ! 1 " 2 @ § # 3 £ ¤ 4 $ % 5 & 6 / 7 { (8 [ ) 9] = 0 } ? + \ É ` ´
Q W E R T Y U I O P Å ^ ü ¨ ~ Enter
Caps Lock A S D F G H J K L Ö Ä * '
> < | Z X C V B N M ; , : . _ -
Reset Ctrl Quit Alt Alt AltGr Enter Ctrl
Dup PA1 FldMk PA2 PgUp ChgSc Jump PgDn PA3
End Insert Delete DelWd
Rule Home
Esc NumLk ScrLk , x Space ÷ Break
7 Home 8 9 Pg Up +
4 5 6 -
1 End 2 3 Pg Dn Enter
0 Ins . Del

122-Key France (251/189) Keyboard

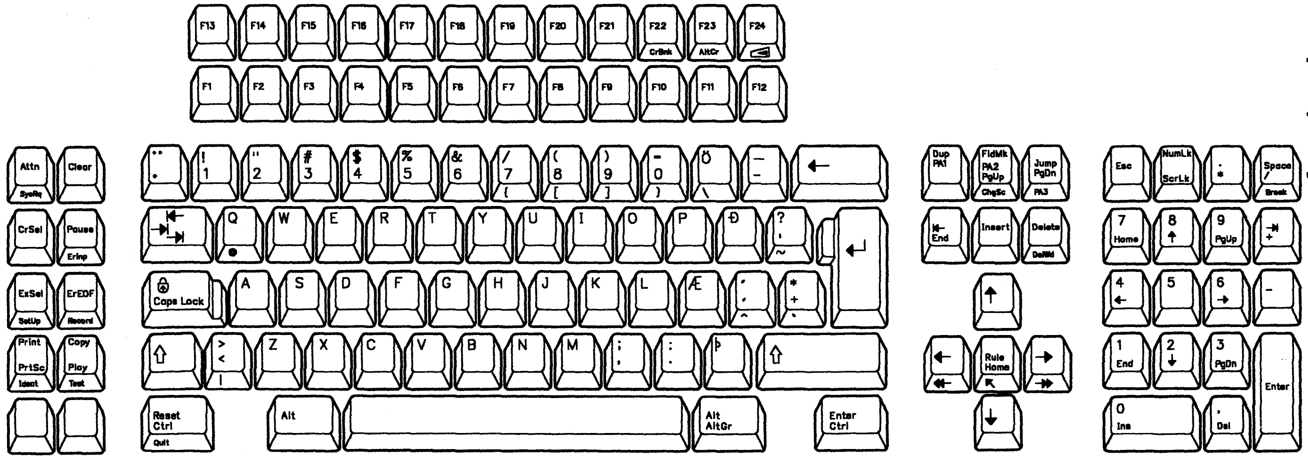


122-Key France (251/120) Keyboard



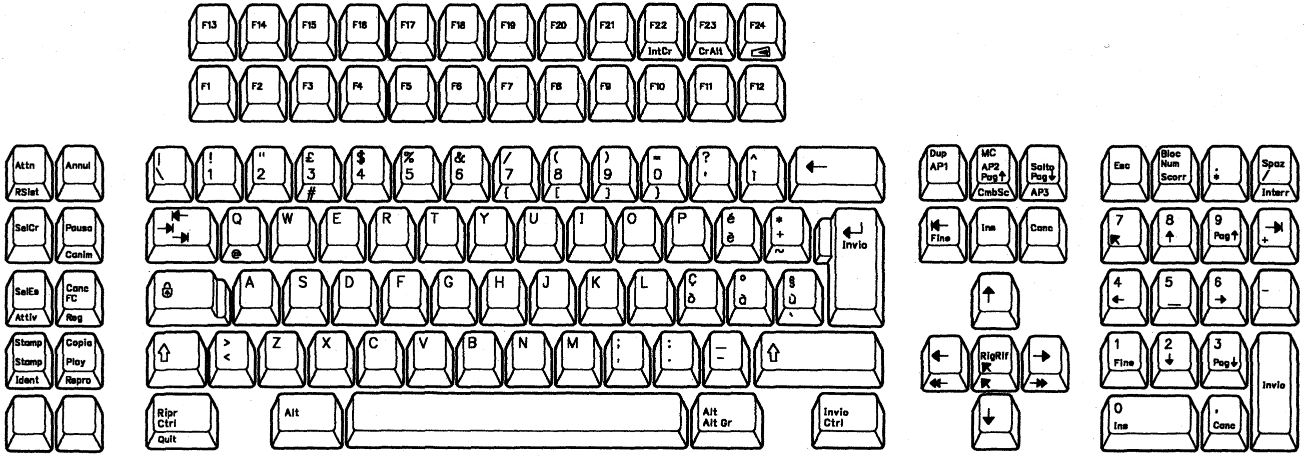

F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 F23 F24
PosBl Pos-U
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Abruf S-Abf Lösch
PaAus Pause E-LS
Er Aus Defin Lösch Feld Aufz
Druck Druck Ident Kop Wdgab Test
° ^ ! 1 " 2 ² § 3 ³ $ 4 % 5 & 6 / 7 { (8 [ ) 9] = 0 } ? ß \ ` ´
Q @ W E R T Z U I O P Ü * + ~
A S D F G H J K L Ö Ä ' #
> < | Y X C V B N M µ ; , : . _ -
Grdst Strg Abbr
Alt
Alt Alt Gr
Daten Freigabe Strg
Dup PA1
FldMk PA2 Bild↑ ÄndBS
Sprg Bild↓ PA3
Ende
Einfg
Entf W-LS
Linie Pos 1
Esc
Num Rollen
. x
Leer ÷ Untbr
7 Pos 1
8 ↑
9 Bild↑
+
4 ←
5
6 →
-
1 Ende
2 ↓
3 Bild↓
Daten Frei-gabe
0 Einfg
, Entf

122-Key Iceland (197) Keyboard

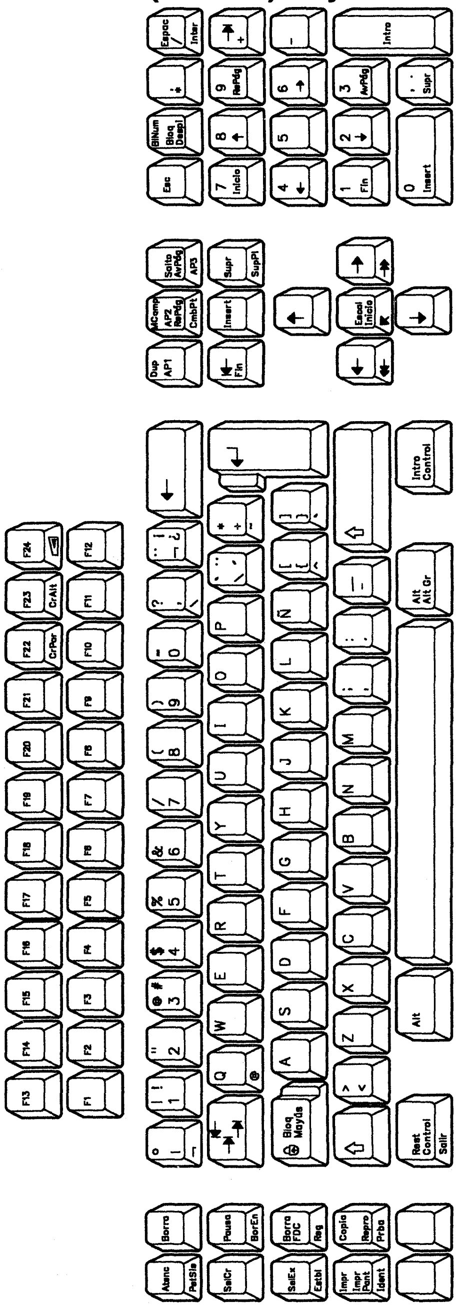


F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 CrBnk F23 AltCr F24
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Attn SysRq Clear
CrSel Pause ErInp
ExSel SetUp ErEOF Record
Print PrtSc Ident Copy Play Test
! 1 " 2 # 3 $ 4 % 5 & 6 / 7 { (8 [ ) 9] = 0 } Ö \ _ -
Q W E R T Y U I O P Đ ? ' ~
Caps Lock A S D F G H J K L Æ ' ^ * + `
> < | Z X C V B N M ; , : . Þ
Reset Ctrl Quit
Alt
Alt AltGr
Enter Ctrl
Dup PA1
FldMk PA2 PgUp ChgSc
Jump PgDn PA3
End
Insert
Delete DelWd
Rule Home
Esc
NumLk ScrLk
Space / Break
7 Home
8
9 PgUp
4
5
6
1 End
2
3 PgDn
Enter
0 Ins
. Del

122-Key Italy (293/142) Keyboard

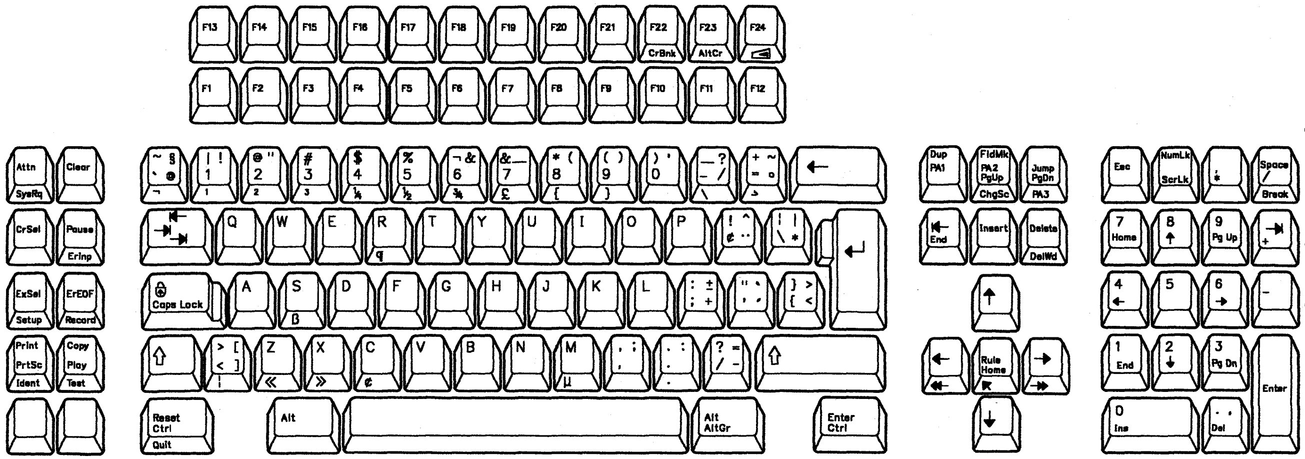

F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 F23 F24
IntCr CrAlt
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Attn RSist Annul
SelCr Pausa Canim
SelEs Attiv Canc FC Reg
Stamp Stamp Ident Copia Play Repro
| \ ! 1 " 2 £ 3 # $ 4 % 5 & 6 / 7 { (8 [ ) 9] = 0 } ? ' ^ ì
Q @ W E R T Y U I O P é è * + ~ Invio
A S D F G H J K L ç ò ° à § ù `
> < Z X C V B N M ; , : . _ -
Ripr Ctrl Quit Alt Alt Alt Gr Invio Ctrl
Dup AP1 MC AP2 Pag CmbSc Salto Pag AP3
Fine Ins Canc
RigRif
Esc Bloc Num Scorr * Spaz / Interr
7 8 9 Pag +
4 5 6 -
1 Fine 2 3 Pag Invio
0 Ins , Canc

122-Key Latin America (309/171) Keyboard



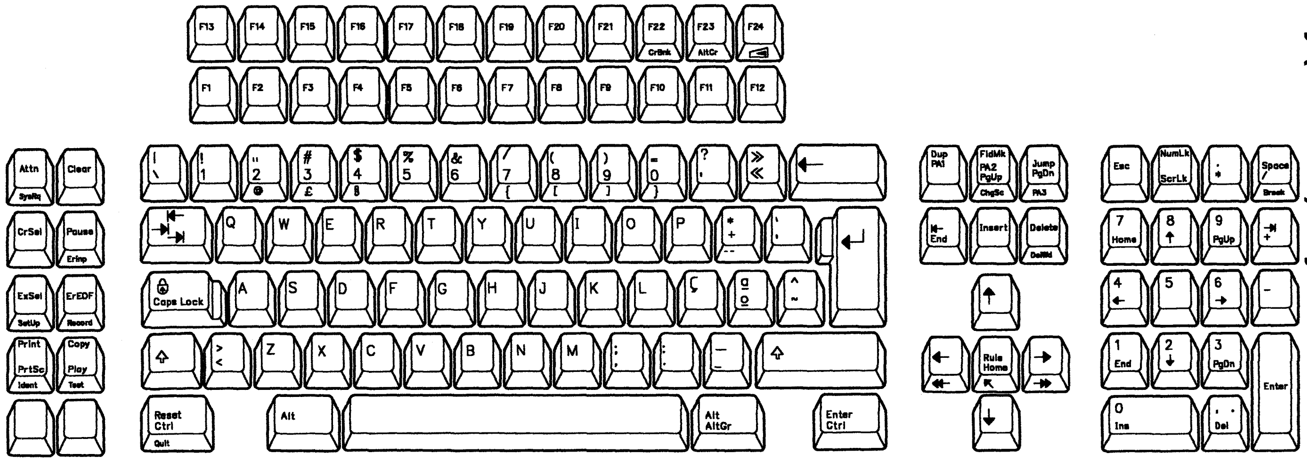
F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 CrPor F23 CrAlt F24
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Atenc PetSis Borro
SelCr Pausa BorEn
SelEx Estbl Borra FDC Reg
Impr Impr Pant Ident Copia Repro Prba
Bloq Mayús
Rest Control Salir
Alt
Alt Alt Gr
Intro Control
Dup AP1 MComp AP2 RePág CmbPt Salto AvPág AP3
Fin Insert Supr SupPl
Escal Inicio
Esc BlNum Bloq Despl Espac Inter
7 Inicio 8 9 RePág
4 5 6
1 Fin 2 3 AvPág Intro
0 Insert Supr

122-Key Netherlands (101/143) Keyboard



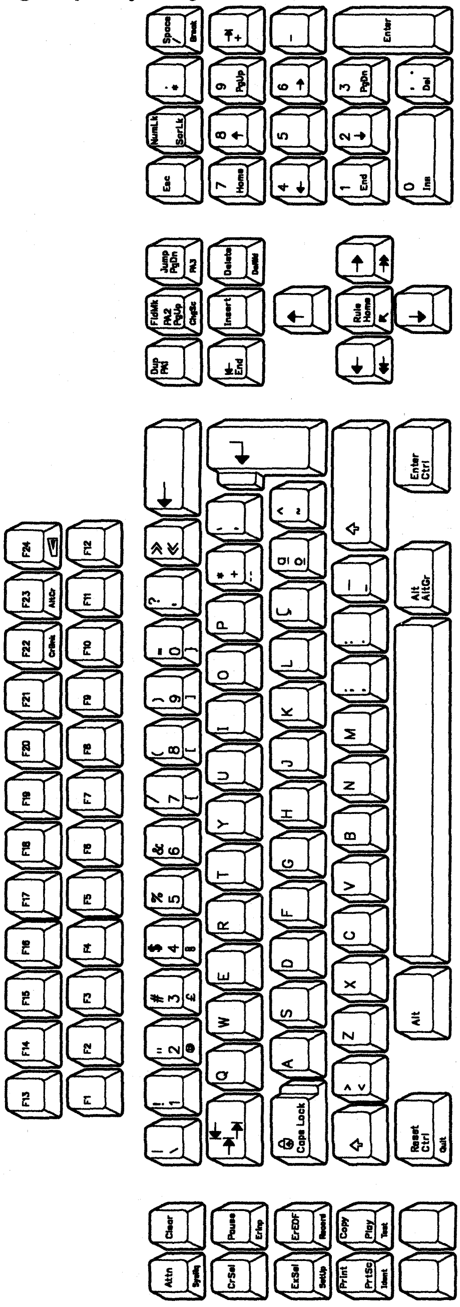
F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 CrBnk F23 AltCr F24
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Attn SysRq Clear
CrSel Pause ErInp
ExSel Setup ErEOF Record
Print PrtSc Ident Copy Play Test
Reset Ctrl Quit
Caps Lock
Alt
Alt AltGr
Enter Ctrl
Dup PA1 FldMk PA2 PgUp ChgSc Jump PgDn PA3
End Insert Delete DelWd
Rule Home
Esc NumLk ScrLk Space / Break
7 Home 8 9 Pg Up
4 5 6
1 End 2 3 Pg Dn Enter
0 Ins . Del

122-Key Norway (281N/155) Keyboard



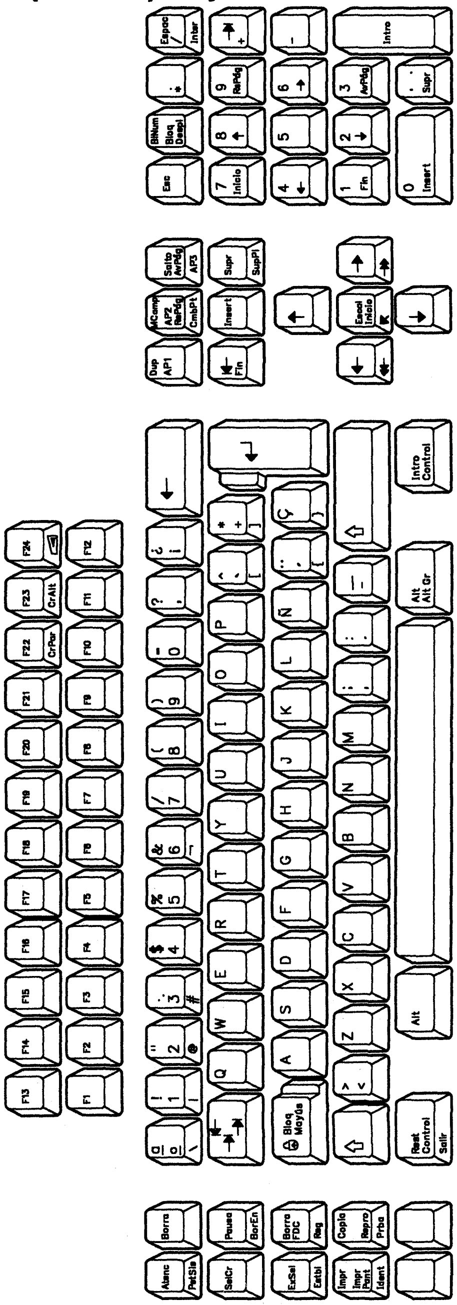
F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 F23 F24
CrBnk AltCr
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Attn SysRq Clear
CrSel Pause ErInp
ExSel SetUp ErEOF Record
Print PrtSc Ident Copy Play Test
| \ ! 1 " 2 @ # 3 £ $ 4 § % 5 & 6 / 7 { (8 [ ) 9] = 0 } ? ' » «
Q W E R T Y U I O P * + ` '
Caps Lock A S D F G H J K L Ç ª º ^ ~
> < Z X C V B N M ; , : . _ -
Reset Ctrl Quit
Alt
Alt AltGr
Enter Ctrl
Dup PA1 FldMk PA2 PgUp ChgSc Jump PgDn PA3
End Insert Delete DelWd
Rule Home
Esc NumLk ScrLk , * Space / Break
7 Home 8 9 PgUp +
4 5 6 -
1 End 2 3 PgDn Enter
0 Ins . , Del

122-Key Portugal (163) Keyboard

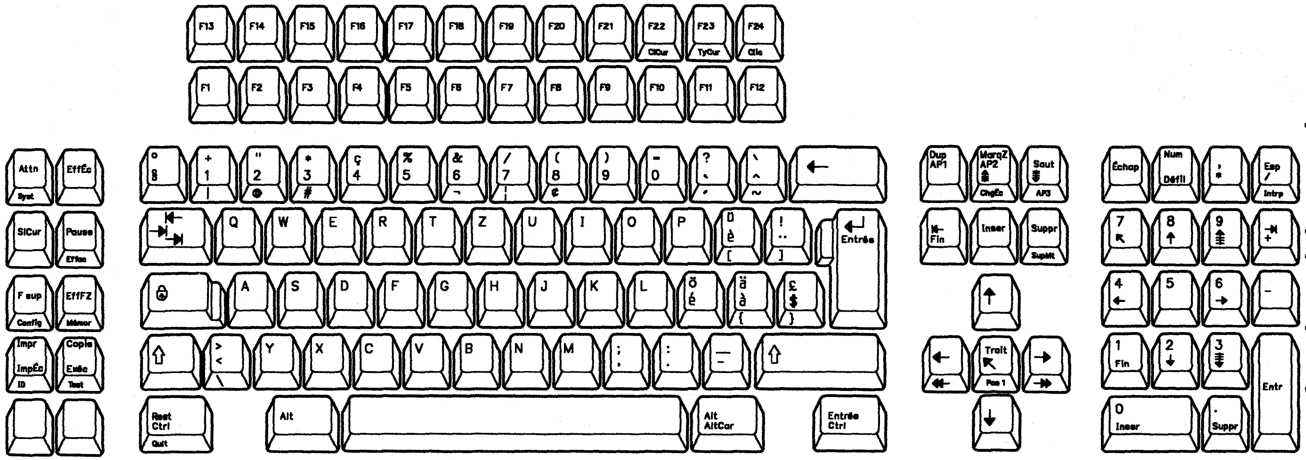


F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 CrBnk F23 AltCr F24
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Attn SysRq Clear
CrSel Pause Erinp
ExSel SetUp ErEOF Record
Print PrtSc Ident Copy Play Test
| \ ! 1 " 2 @ # 3 £ $ 4 § % 5 & 6 / 7 { (8 [ ) 9] = 0 } ? ' » «
Q W E R T Y U I O P * + ¨ ´ `
Caps Lock A S D F G H J K L Ç ª º ^ ~
> < Z X C V B N M ; , : . _ -
Reset Ctrl Quit Alt Alt AltGr Enter Ctrl
Dup PA1 FldMk PA2 PgUp ChgSc Jump PgDn PA3
End Insert Delete DelWd
Rule Home
Esc NumLk ScrLk . * Space / Break
7 Home 8 9 PgUp +
4 5 6 -
1 End 2 3 PgDn Enter
0 Ins . , Del

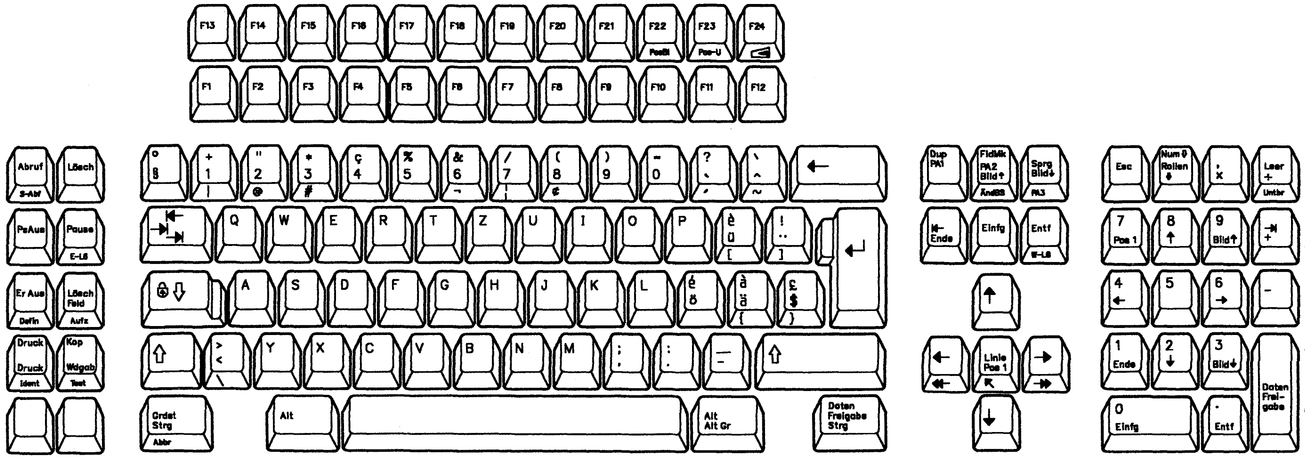
122-Key Spain (071/172) Keyboard



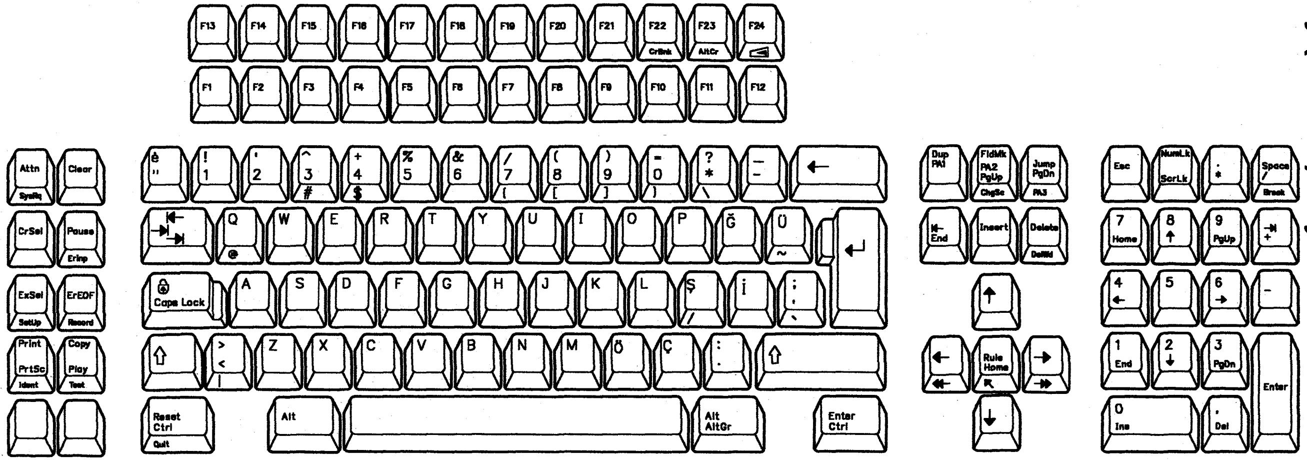
122-Key Switzerland (French) (150F) Keyboard



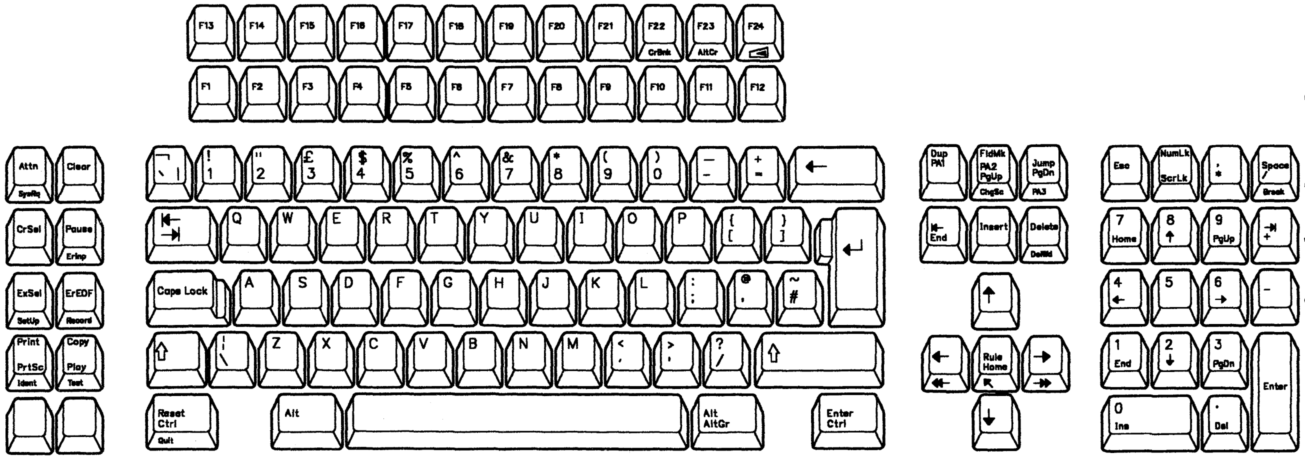
122-Key Switzerland (German) (150G) Keyboard


F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 F23 F24
PosBl Pos-U
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Abruf S-Abf Lösch
PsAus Pause E-LS
Er Aus Defin Lösch Feld Aufz
Druck Druck Ident Kop Wdgab Test
° § + 1 | " 2 @ * 3 # ç 4 % 5 & 6 ¬ / 7 ¦ (8 ¢) 9 = 0 ? ' ´ ` ^ ~
Q W E R T Z U I O P è ü [! ¨]
A S D F G H J K L é ö à ä { £ $ }
> < \ Y X C V B N M ; , : . _ -
Grdst Strg Abbr
Alt
Alt Alt Gr
Daten Freigabe Strg
Dup PA1
FldMk PA2 Bild↑ ÄndBS
Sprg Bild↓ PA3
Ende
Einfg
Entf W-LS
Linie Pos 1
Esc
Num Rollen
, x
Leer + Untbr
7 Pos 1 8 ↑ 9 Bild↑
4 ← 5 6 → -
1 Ende 2 ↓ 3 Bild↓
Daten Frei- gabe
0 Einfg
. Entf

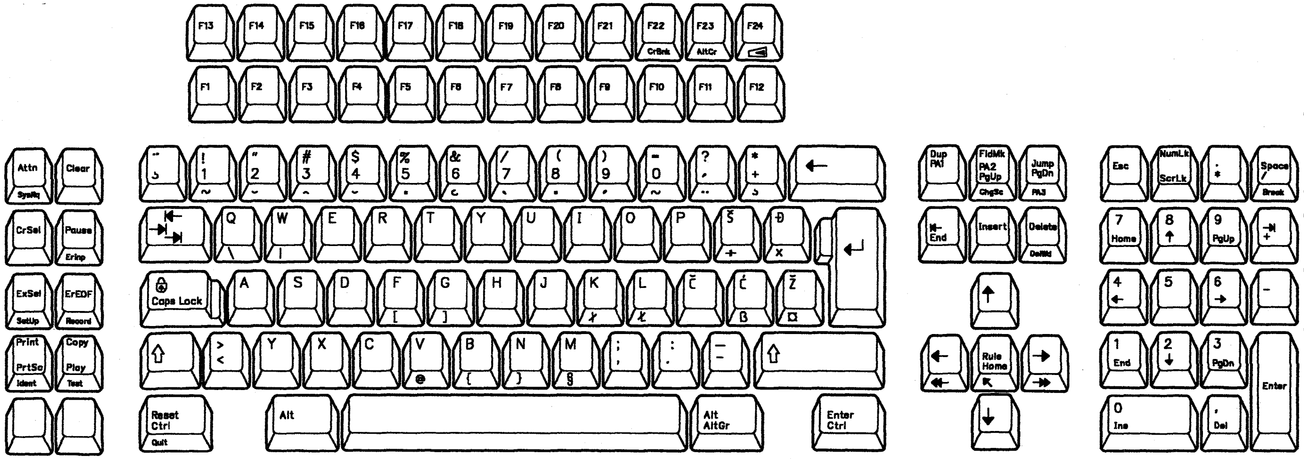
122-Key Turkey (402/179) Keyboard


F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 F23 F24
CrBnk AltCr
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Attn SysRq Clear
CrSel Pause ErInp
ExSel SetUp ErEOF Record
Print PrtSc Ident Copy Play Test
é " ! 1 ' 2 ^ 3 # + 4 $ % 5 & 6 / 7 { (8 [ ) 9] = 0 } ? * \ _ -
Q @ W E R T Y U I O P Ğ Ü ~
Caps Lock A S D F G H J K L Ş / İ ; ' `
> < | Z X C V B N M Ö Ç : .
Reset Ctrl Quit
Alt
Alt AltGr
Enter Ctrl
Dup PA1 FldMk PA2 PgUp ChgSc Jump PgDn PA3
End Insert Delete DelWd
Rule Home
Esc NumLk ScrLk * Space / Break
7 Home 8 9 PgUp +
4 5 6 -
1 End 2 3 PgDn Enter
0 Ins , Del

122-Key United Kingdom (166) Keyboard



122-Key Yugoslavia (Latin) (234) Keyboard



Index

Characters and Keystrokes

Notes:

Character Codes

The following figures show the decimal values, hexadecimal values, and keystrokes for each character. The notes referred to in the figures are on page 7.

Value		As Characters		
Hex	Dec	Symbol	Keystrokes	Notes
00	0	Blank (Null)	Ctrl 2	
01	1	☺	Ctrl A	
02	2	☻	Ctrl B	
03	3	♥	Ctrl C	
04	4	♦	Ctrl D	
05	5	♣	Ctrl E	
06	6	♠	Ctrl F	
07	7	•	Ctrl G	
08	8	◘	Ctrl H, Backspace, Shift Backspace	
09	9	○	Ctrl I	
0A	10	◙	Ctrl J, Ctrl ↵	
0B	11	♂	Ctrl K	
0C	12	♀	Ctrl L	
0D	13	♪	Ctrl M, ↵, Shift ↵	
0E	14	♫	Ctrl N	
0F	15	☼	Ctrl O	
10	16	►	Ctrl P	
11	17	◄	Ctrl Q	
12	18	↕	Ctrl R	
13	19	‼	Ctrl S	
14	20	¶	Ctrl T	
15	21	§	Ctrl U	
16	22	▬	Ctrl V	
17	23	↨	Ctrl W	

Value		As Characters		
Hex	Dec	Symbol	Keystrokes	Notes
18	24	↑	Ctrl X	
19	25	↓	Ctrl Y	
1A	26	→	Ctrl Z	
1B	27	←	Ctrl [Esc, Shift Esc, Crtl Esc	
1C	28	∟	Ctrl	
1D	29	↔	Ctrl]	
1E	30	▲	Ctrl 6	
1F	31	▼	Ctrl -	
20	32	Blank Space	Space Bar, Shift, Space, Ctrl Space, Alt Space	
21	33	!	!	Shift
22	34	"	"	Shift
23	35	#	#	Shift
24	36	$	$	Shift
25	37	%	%	Shift
26	38	&	&	Shift
27	39	'	'	Shift
28	40	(	(	Shift
29	41	)	)	
2A	42	*	*	Note 1
2B	43	+	+	Shift
2C	44	,	,	
2D	45	-	-	
2E	46	.	.	Note 2

Value		As Characters		
Hex	Dec	Symbol	Keystrokes	Notes
2F	47	/	/	
30	48	0	0	Note 3
31	49	1	1	Note 3
32	50	2	2	Note 3
33	51	3	3	Note 3
34	52	4	4	Note 3
35	53	5	5	Note 3
36	54	6	6	Note 3
37	55	7	7	Note 3
38	56	8	8	Note 3
39	57	9	9	Note 3
3A	58	:	:	Shift
3B	59	;	;	
3C	60	<	<	Shift
3D	61	=	=	
3E	62	>	>	Shift
3F	63	?	?	Shift
40	64	@	@	Shift
41	65	A	A	Note 4
42	66	B	B	Note 4
43	67	C	C	Note 4
44	68	D	D	Note 4
45	69	E	E	Note 4
46	70	F	F	Note 4
47	71	G	G	Note 4
48	72	H	H	Note 4
49	73	I	I	Note 4
4A	74	J	J	Note 4
4B	75	K	K	Note 4
4C	76	L	L	Note 4
4D	77	M	M	Note 4
4E	78	N	N	
4F	79	O	O	Note 4
50	80	P	P	Note 4
51	81	Q	Q	Note 4
52	82	R	R	Note 4
53	83	S	S	Note 4
54	84	T	T	Note 4
55	85	U	U	Note 4
56	86	V	V	Note 4
57	87	W	W	Note 4
58	88	X	X	Note 4
59	89	Y	Y	Note 4
5A	90	Z	Z	Note 4
5B	91	[	[	
5C	92	\	\	Note 4
5D	93	]	]	
5E	94	^	^	Shift
5F	95	_	_	Shift
60	96	`	`	
61	97	a	a	Note 5
62	98	b	b	Note 5
63	99	c	c	Note 5
64	100	d	d	Note 5
65	101	e	e	Note 5
66	102	f	f	Note 5

Value		As Characters		
Hex	Dec	Symbol	Keystrokes	Notes
67	103	g	g	Note 5
68	104	h	h	Note 5
69	105	i	i	Note 5
6A	106	j	j	Note 5
6B	107	k	k	Note 5
6C	108	l	l	Note 5
6D	109	m	m	Note 5
6E	110	n	n	Note 5
6F	111	o	o	Note 5
70	112	p	p	Note 5
71	113	q	q	Note 5
72	114	r	r	Note 5
73	115	s	s	Note 5
74	116	t	t	Note 5
75	117	u	u	Note 5
76	118	v	v	Note 5
77	119	w	w	Note 5
78	120	x	x	Note 5
79	121	y	y	Note 5
7A	122	z	z	Note 5
7B	123	{	{	Shift
7C	124	¦	¦	Shift
7D	125	}	}	Shift
7E	126	~	~	Shift
7F	127	△	Ctrl-	

Value		As Characters		
Hex	Dec	Symbol	Keystrokes	Notes
80	128	Ç	Alt 128	Note 6
81	129	ü	Alt 129	Note 6
82	130	é	Alt 130	Note 6
83	131	â	Alt 131	Note 6
84	132	ä	Alt 132	Note 6
85	133	à	Alt 133	Note 6
86	134	å	Alt 134	Note 6
87	135	ç	Alt 135	Note 6
88	136	ê	Alt 136	Note 6
89	137	ë	Alt 137	Note 6
8A	138	è	Alt 138	Note 6
8B	139	ï	Alt 139	Note 6
8C	140	î	Alt 140	Note 6
8D	141	ì	Alt 141	Note 6
8E	142	Ä	Alt 142	Note 6
8F	143	Å	Alt 143	Note 6
90	144	É	Alt 144	Note 6
91	145	æ	Alt 145	Note 6
92	146	Æ	Alt 146	Note 6
93	147	ô	Alt 147	Note 6
94	148	ö	Alt 148	Note 6
95	149	ò	Alt 149	Note 6
96	150	û	Alt 150	Note 6
97	151	ù	Alt 151	Note 6
98	152	ÿ	Alt 152	Note 6
99	153	Ö	Alt 153	Note 6
9A	154	Ü	Alt 154	Note 6

Value		As Characters		
Hex	Dec	Symbol	Keystrokes	Notes
9B	155	¢	Alt 155	Note 6
9C	156	£	Alt 156	Note 6
9D	157	¥	Alt 157	Note 6
9E	158	Pt	Alt 158	Note 6
9F	159	ƒ	Alt 159	Note 6
A0	160	á	Alt 160	Note 6
A1	161	í	Alt 161	Note 6
A2	162	ó	Alt 162	Note 6
A3	163	ú	Alt 163	Note 6
A4	164	ñ	Alt 164	Note 6
A5	165	Ñ	Alt 165	Note 6
A6	166	ª	Alt 166	Note 6
A7	167	º	Alt 167	Note 6
A8	168	¿	Alt 168	Note 6
A9	169	⌐	Alt 169	Note 6
AA	170	¬	Alt 170	Note 6
AB	171	½	Alt 171	Note 6
AC	172	¼	Alt 172	Note 6
AD	173	¡	Alt 173	Note 6
AE	174	<<	Alt 174	Note 6
AF	175	>>	Alt 175	Note 6
B0	176	░	Alt 176	Note 6
B1	177	▒	Alt 177	Note 6
B2	178	▓	Alt 178	Note 6
B3	179	│	Alt 179	Note 6
B4	180	┤	Alt 180	Note 6
B5	181	╡	Alt 181	Note 6
B6	182	╢	Alt 182	Note 6
B7	183	╖	Alt 183	Note 6
B8	184	╕	Alt 184	Note 6
B9	185	╣	Alt 185	Note 6
BA	186	║	Alt 186	Note 6
BB	187	╗	Alt 187	Note 6
BC	188	╝	Alt 188	Note 6
BD	189	╜	Alt 189	Note 6
BE	190	╛	Alt 190	Note 6
BF	191	┐	Alt 191	Note 6
C0	192	└	Alt 192	Note 6
C1	193	┴	Alt 193	Note 6
C2	194	┬	Alt 194	Note 6
C3	195	├	Alt 195	Note 6
C4	196	─	Alt 196	Note 6
C5	197	┼	Alt 197	Note 6
C6	198	╞	Alt 198	Note 6
C7	199	╟	Alt 199	Note 6
C8	200	╚	Alt 200	Note 6
C9	201	╔	Alt 201	Note 6
CA	202	╩	Alt 202	Note 6
CB	203	╦	Alt 203	Note 6
CC	204	╠	Alt 204	Note 6
CD	205	═	Alt 205	Note 6
CE	206	╬	Alt 206	Note 6
CF	207	╧	Alt 207	Note 6
D0	208	╨	Alt 208	Note 6

Value		As Characters		
Hex	Dec	Symbol	Keystrokes	Notes
D1	209	╤	Alt 209	Note 6
D2	210	╥	Alt 210	Note 6
D3	211	╙	Alt 211	Note 6
D4	212	╘	Alt 212	Note 6
D5	213	╒	Alt 213	Note 6
D6	214	╓	Alt 214	Note 6
D7	215	╫	Alt 215	Note 6
D8	216	╪	Alt 216	Note 6
D9	217	┘	Alt 217	Note 6
DA	218	┌	Alt 218	Note 6
DB	219	█	Alt 219	Note 6
DC	220	▄	Alt 220	Note 6
DD	221	▌	Alt 221	Note 6
DE	222	▐	Alt 222	Note 6
DF	223	▀	Alt 223	Note 6
EO	224	α	Alt 224	Note 6
E1	225	β	Alt 225	Note 6
E2	226	Γ	Alt 226	Note 6
E3	227	π	Alt 227	Note 6
E4	228	Σ	Alt 228	Note 6
E5	229	σ	Alt 229	Note 6
E6	230	μ	Alt 230	Note 6
E7	231	τ	Alt 231	Note 6
E8	232	Φ	Alt 232	Note 6
E9	233	θ	Alt 233	Note 6
EA	234	Ω	Alt 234	Note 6
EB	235	δ	Alt 235	Note 6
EC	236	∞	Alt 236	Note 6
ED	237	φ	Alt 237	Note 6
EE	238	ε	Alt 238	Note 6
EF	239	∩	Alt 239	Note 6
F0	240	≡	Alt 240	Note 6
F1	241	±	Alt 241	Note 6
F2	242	≥	Alt 242	Note 6
F3	243	≤	Alt 243	Note 6
F4	244	⌠	Alt 244	Note 6
F5	245	⌡	Alt 245	Note 6
F6	246	÷	Alt 246	Note 6
F7	247	≈	Alt 247	Note 6
F8	248	°	Alt 248	Note 6
F9	249	●	Alt 249	Note 6
FA	250	•	Alt 250	Note 6
FB	251	√	Alt 251	Note 6
FC	252	^{n}	Alt 252	Note 6
FD	253	^{2}	Alt 253	Note 6
FE	254	■	Alt 254	Note 6
FF	255	BLANK	Alt 255	Note 6

Notes:

1. Asterisk (*) can be typed by pressing the * key or, in the shift state, pressing the 8 key.
2. Period (.) can be typed by pressing the . key or, in the shift or Num Lock state, pressing the Del key.
3. Numeric characters 0-9 can be typed by pressing the numeric keys on the keyboard or, in the shift or Num Lock state, pressing the numeric keys in the keypad portion of the keyboard.
4. Uppercase alphabetic characters (A-Z) can be typed by pressing the character key in the shift state or the Caps Lock state.
5. Lowercase alphabetic characters (a-z) can be typed by pressing the character key in the normal state or in Caps Lock and shift state combined.
6. The three digits are typed on the numeric keypad while pressing the Alt key. Character codes 001-255 may be entered in this fashion (with Caps Lock activated, character codes 97-122 display in uppercase).

Quick Reference

DECIMAL VALUE	➡	0	16	32	48	64	80	96	112
⬇	HEXA-DECIMAL VALUE	0	1	2	3	4	5	6	7
0	0	BLANK (NULL)	►	BLANK (SPACE)	0	@	P	‘	p
1	1	☺	◄	!	1	A	Q	a	q
2	2	☻	↕	"	2	B	R	b	r
3	3	♥	‼	#	3	C	S	c	s
4	4	♦	¶	$	4	D	T	d	t
5	5	♣	§	%	5	E	U	e	u
6	6	♠	▬	&	6	F	V	f	v
7	7	•	↨	′	7	G	W	g	w
8	8	◘	↑	(	8	H	X	h	x
9	9	○	↓	)	9	I	Y	i	y
10	A	◙	→	*	:	J	Z	j	z
11	B	♂	←	+	;	K	[	k	{
12	C	♀	∟	,	<	L	\	l	¦
13	D	♪	↔	—	=	M	]	m	}
14	E	♫	▲	.	>	N	^	n	~
15	F	☼	▼	/	?	O	_	o	△

DECIMAL VALUE	➡	128	144	160	176	192	208	224	240
⬇	HEXA-DECIMAL VALUE	8	9	A	B	C	D	E	F
0	0	Ç	É	á	░	└	╨	∝	≡
1	1	ü	æ	í	▒	┴	╤	β	±
2	2	é	Æ	ó	▓	┬	╥	Γ	≥
3	3	â	Ô	ú	│	├	╙	π	≤
4	4	ä	ö	ñ	┤	─	╘	Σ	⌠
5	5	à	ò	Ñ	╡	┼	╒	σ	⌡
6	6	å	û	ª	╢	╞	╓	μ	÷
7	7	ç	ù	º	╖	╟	╫	ϒ	≈
8	8	ê	ÿ	¿	╕	╚	╪	Φ	°
9	9	ë	Ö	⌐	╣	╔	┘	Θ	•
10	A	è	Ü	¬	║	╩	┌	Ω	•
11	B	ï	¢	½	╗	╦	█	δ	√
12	C	î	£	¼	╝	╠	▄	∞	n
13	D	ì	¥	¡	╜	═	▌	φ	2
14	E	Ä	₧	«	╛	╬	▐	∈	■
15	F	Å	ƒ	»	┐	╧	▀	∩	BLANK 'FF'

Notes:

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.

	Software Written for:		
	8087	**80287**	**80287**
Operating Modes	**Real**	**Real**	**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.

SCSI Subsystem

Figures

Description

The SCSI subsystem serves as the interface between the system and devices using Small Computer System Interface (SCSI) architecture.

The subsystem supports:

- A broad range of internal and external SCSI devices.
- Up to seven SCSI physical devices. Each physical device can support eight logical devices.
- Overlapped command processing for up to 15 devices.
- Devices compatible with SCSI Common Command Set.
- A data transfer rate of up to 5 million bytes-per-second between the subsystem and the external devices.
- A data transfer rate of up to 20 million bytes-per-second from the subsystem to the system (burst rate).
- A 16-bit data bus.
- A 24- or 32-bit address bus (automatically configurable).

The subsystem also serves as a bus master (intelligent bus controller).

Warning: In this technical reference, the term "Reserved" describes certain signals, bits, and registers that should not be changed. Use of reserved areas can cause compatibility problems, loss of data, or permanent damage to the hardware. When the contents of a register are changed, the state of the reserved bits must be preserved. When possible, read the register first and change only the bits that must be changed.

The following is a block diagram of the subsystem.

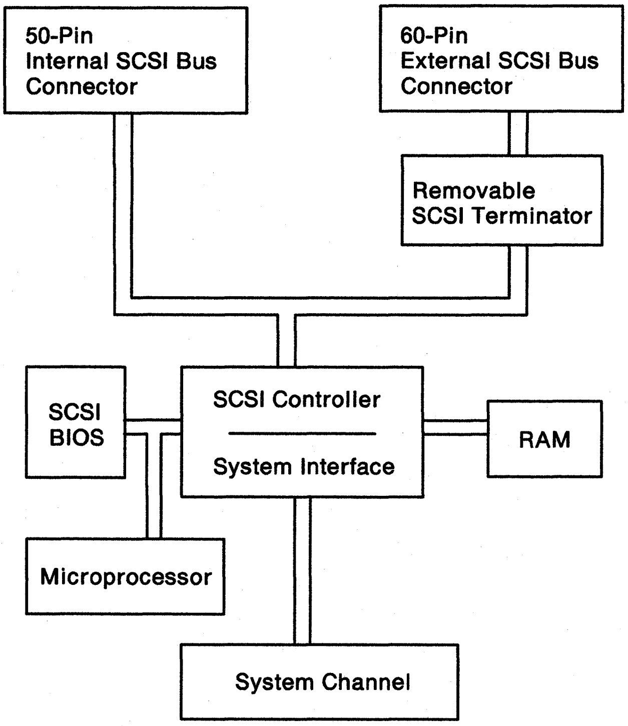

50-Pin Internal SCSI Bus Connector
60-Pin External SCSI Bus Connector
Removable SCSI Terminator
SCSI BIOS
SCSI Controller
System Interface
RAM
Microprocessor
System Channel

Figure 1. Subsystem Block Diagram

Subsystem Components

The subsystem components described in this section are:

- Microprocessor
- SCSI bus controller
- Data flow controller
- System interface/bus master controller.

Microprocessor

The subsystem microprocessor controls all subsystem operations. It translates commands received from the system into a series of operations. For example, the subsystem microprocessor controls data transfers to and from the system, and executes all necessary error detection and recovery procedures to ensure data integrity.

SCSI Bus Controller

The SCSI bus controller transmits commands, receives status, transfers data between attached SCSI devices and the subsystem, and provides the following functions:

- SCSI bus arbitration
- Device selection and reselection
- SCSI phase change detection
- SCSI bus parity generation and checking.

Data Flow Controller

The data flow controller manages the flow of parallel data between the SCSI bus and the system. The controller enables concurrent data transfers between the SCSI bus and the system.

The bus-steering and data-flow logic enables either byte or word transfers to the system, and performs these transfers in the pipeline or nonpipeline mode.

System Interface/Bus Master Controller

The system interface/bus master controller provides the command and interrupt registers. These registers receive commands from the system and interrupt the system when a command is completed. The transfer of data between the subsystem and the system is controlled through the bus master. Data transfers of up to 20 million bytes-per-second are supported.

Statement of Conformance

The following statements are in the format specified in Appendix E of the American National Standards Institute (ANSI) SCSI Standard X3.131-1986.

Alternatives

- Single-ended receivers and drivers are used.
- +Terminator Power is supplied by the system board.
- Parity is used on all SCSI data transfers.
- Hard reset of the SCSI bus is implemented.
- Target reservation queueing is supported.

Level of Conformance

The level of conformance is Level 2.

Implemented Options

The Synchronous Data Transfer Request message is supported. Maximum request-to-acknowledge offset is 7. Minimum transfer period implemented is 200 nanoseconds.

Supported Devices

The subsystem supports devices that conform to the ANSI SCSI Standard X3.131-1986. These devices must support the mandatory commands and messages specified in Addendum 4.B, also known as the Common Command Set.

Each SCSI device connected to a single SCSI subsystem must be given, at time of installation, a unique SCSI identification number (ID). The ID must be set on each device at the time of installation. The ID must not be the same ID given to any other device connected to the same subsystem, or the ID given to the subsystem itself. The SCSI ID is also the physical unit number (PUN).

Supported SCSI Commands

The subsystem supports the following SCSI commands from the Common Command Set.

Copy *	Release
Format Unit *	Request Sense *
Inquiry *	Reserve
Mode Select	Rezero Unit
Mode Sense	Seek
Prevent/Allow Medium Removal	Send Diagnostic
Read *	Start/Stop Unit
Read Buffer	Test Unit Ready
Read Capacity *	Verify *
Read Defect Data	Write *
Read Extended *	Write and Verify *
Reassign Blocks *	Write Buffer
Receive Diagnostic Results	Write Extended *

* SCSI commands issued by the controller when translating the system interface commands (op codes hex 01 to 1E).

Note: SCSI commands shown without an asterisk can be issued only with the command Send Other SCSI Command. SCSI commands not shown are not supported.

Figure 2. SCSI Commands

Supported SCSI Messages

The following SCSI messages are supported by the subsystem.

SCSI Messages	**SCSI Status Messages**
Abort	Busy
Bus-Device Reset	Check Condition
Command Complete	Condition Met/Good
Disconnect	Good
Identify	Immediate/Good
Initiator-Detected Error	Intermediate/Condition Met/Good
Linked Command Complete	Reservation Conflict
Linked Command Complete with Flag	
Message Parity Error	
Message Reject	
No Operation	
Restore Pointers	
Save Data-Pointer	

Note: The SCSI extended message Synchronous Data Transfer Request is also supported by the subsystem.

Figure 3. SCSI Messages

Device Requirements for POST Support

The following are the minimum requirements that SCSI devices must meet to be supported by the POST routines.

- All devices must be able to respond to the Inquiry and Request Sense commands within 3 seconds after being powered-on.
- Device type 0 must respond to the Start/Stop Unit and Test Unit Ready commands within 3 seconds after being powered-on.
- Device type 0 with non-removable media must make the drive ready for media access within 30 seconds of receiving a Start Unit command.
- Device type 0 must respond to the Read and Read Capacity commands.
- While configuring the SCSI bus, POST issues the Inquiry command to all 56 possible combinations of physical unit number/logical unit number (PUN/LUN) until 15 logical devices are assigned. Therefore, all devices must respond to each possible LUN for its SCSI ID.

Programmable Option Select

The subsystem contains options that can be configured. These options are controlled through the programmable option select (POS) registers. These registers are accessible only when the 'chip setup' signal is active.

Warning: IBM recommends that programmable options be set only through System Configuration utilities. Directly setting the complementary metal-oxide semiconductor (CMOS) parameters or POS registers can result in multiple assignment of the same system resource, improper operation of the feature, loss of data, or possible damage to the hardware.

Address (Hex)	Register
0102	POS Register 2
0103	POS Register 3

Figure 4. POS Register Selection

Note: When the system is modifying a register, the state of the reserved bits must be preserved. When possible, read the register first and change only the bits required.

POS Register 2 (Hex 0102)

Bit	Function
7 − 4	Revision ID
3 − 2	Reserved
1	8K NVRAM Disabled
0	Chip Enable

Figure 5. POS Register 2 (Hex 0102)

Bits 7 − 4 These read-only bits identify the level of the chip.

Bits 3 − 2 These bits are reserved.

Bit 1 Setting this bit to 1 blocks access to the 8K NVRAM. Do not set this bit to 1 because this function is unsupported by the system.

Bit 0 When this bit is set to 1, the chip is enabled. When this bit is set to 0, the chip responds only to POS read and write operations. Subsystem interrupts are disabled. This bit is set to 0 when the ‘channel reset’ signal on the system channel becomes active.

POS Register 3 (Hex 0103)

Bit	Function
7−5	SCSI ID
4	Reserved = 0
3−0	Reserved = 0

Figure 6. POS Register 3 (Hex 0103)

Bits 7−5 These bits determine the SCSI ID (0 through 7) of the subsystem. Each device on the SCSI bus must have a unique SCSI ID. If the SCSI ID for the subsystem is changed, a subsystem-hardware reset should be performed.

Bit 4 This bit is reserved and must be set to 0.

Bits 3−0 These bits are reserved and must be set to 0.

Subsystem-to-System Interface

The subsystem operates as a bus master to the system in a subsystem-to-system interface mode. A set of eight I/O registers issues commands and receives status from the subsystem. The addresses for these registers are 3540 hex through 3547 hex.

Register	Read/Write	Offset
Command Interface Register 1	Read/Write	00
Command Interface Register 2	Read/Write	01
Command Interface Register 3	Read/Write	02
Command Interface Register 4	Read/Write	03
Attention Register	Read/Write	04
Basic Control Register	Read/Write	05
Interrupt Status Register	Read	06
Basic Status Register	Read	07

Figure 7. I/O Registers

Command Interface Registers (Hex 00−03)

The system uses these four 8-bit registers to transfer either a 32-bit immediate command or the subsystem control block (SCB) address from the system to the subsystem. The immediate command or the SCB specifies the operation to be performed by the subsystem and also specifies any necessary parameters.

Before the system writes an attention request code (1, 3, 4, or F) and a device number to the Attention register, it writes an immediate command or the SCB address to the Command Interface registers. The subsystem interrupts the system when the command is completed.

The following figures show the relationship between the Command Interface registers and the immediate command or the SCB address.

```
               Command Interface Register Bits
 15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0

  <-- Command Register 1 ---> < Command Register 0 >    Low SCB Address

  <-- Command Register 3 ---> < Command Register 2 >    High SCB Address
```

Figure 8. Register Content for SCB

```
               Command Interface Register Bits
 15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0

  <-- Command Register 1 ---> < Command Register 0 >    Word 0

  <-- Command Register 3 ---> < Command Register 2 >    Word 1
```

Figure 9. Register Content for Immediate Command

Attention Register (Hex 04)

The system uses this 8-bit read/write register to request a subsystem operation. The value in this register specifies the operation and the device. When a value is written to the Attention register, bit 0 (busy) in the Basic Status register is set to 1. The busy bit remains set until the subsystem is ready for another attention request. This bit is usually cleared within approximately 50 microseconds; however, if a Reset command to the subsystem is received, the busy bit remains set until the subsystem microprocessor has completed the reset sequence (up to 30 seconds).

With system interrupts disabled, the system must verify that the busy bit is set to 0 before writing to the command interface registers or the Attention register.

Bit	Name
7–4	Request Code
3–0	Device Select Code

Figure 10. Attention Register

Bits 7–4 The request code prompts the subsystem to perform a specific action for the indicated device or subsystem. The following figure provides additional information regarding request codes.

Hex Value	Description
1	Immediate Command
3	Start SCB
4	Start Long SCB
E	End of Interrupt
F	Start Long SCB

Figure 11. Request Codes

1 Requests that the subsystem perform the immediate command specified in the Command Interface registers.

3 Requests that the subsystem perform the command, as indicated by the SCB in system memory, at the 24-bit address contained in the command interface registers.

4	Requests that the subsystem perform the command as indicated by the long SCB in system memory. This request also can be used to start a Send Other SCSI Command SCB. If started with this code, the entire Send Other SCSI Command SCB will be fetched at the same time. If a Send Other SCSI Command SCB is started by request code 3, two control-block fetches are required to get the entire command.
E	Indicates to the subsystem that the system has completed processing the last interrupt. The subsystem can issue another interrupt to the system and reset the hardware interrupt-request latch. The end-of-interrupt (EOI) request is not the same as the EOI instruction to the interrupt controller. The EOI instruction is in addition to the EOI request code required by the subsystem to properly end an interrupt. To avoid false interrupts, issue the EOI request to the subsystem before issuing the EOI to the system interrupt controller. The subsystem requires 250 nanoseconds to clear the interrupt request after issuing the EOI request.
F	Same as 4.

The following information refers to Figure 10 on page 12.

Bits 3 – 0	These bits specify the number of the logical device that will perform the operation. The value of hex F specifies the subsystem and is used for global commands. The system can change the number assigned to a SCSI device through the Assign command. When the subsystem is reset, the device assignments are set to their default values. For further information, refer to Figure 15 on page 16.

Basic Control Register (Hex 05)

This 8-bit read/write register controls various subsystem functions.

Bit	Symbol
7	Hardware Reset
6−2	Reserved
1	Direct Memory Access (DMA) Enable
0	Interrupt Enable

Figure 12. Basic Control Register

Bit 7 When this bit is set to 1, the subsystem hardware is reset. The subsystem stays in the reset mode until this bit is set to 0. This reset does not affect other devices on the SCSI bus. POS information is not affected. When this bit is set to 1, the system should not set this bit to 0 for at least 50 microseconds. This bit must be set to 0 before setting either bit 0 or bit 1 in this register to 1.

Bits 6−2 These bits are reserved.

Bit 1 When set to 1, this bit enables the subsystem controller.

Bit 0 When set to 1, this bit enables interrupts to the system.

Interrupt Status Register (Hex 06)

The subsystem uses this 8-bit read-only register to return command-completion information to the system. If the subsystem interrupt is enabled, a system interrupt (IRQ14) is generated when the subsystem updates the Interrupt Status register.

After receiving an interrupt, the system may read the Interrupt Status register to determine the device and ID of the interrupt. The system may clear the interrupt by issuing an EOI request. This register also is updated after the subsystem is reset. For more information see "Resetting the Subsystem" on page 18.

The subsystem clears the Interrupt Status register only after an end-of-interrupt request. This enables the program to read the Interrupt Status register as many times as necessary to determine the routine that will process the interrupt.

Bit	Symbol
7−4	Interrupt ID
3−0	Device ID Number

Figure 13. Interrupt Status Register

Bits 7 – 4 These bits identify the type of interrupt, as shown in the following figure.

More information about interrupt status is available in the command status block (see "Get Command Complete Status" on page 34).

Hex	Interrupt
1	SCB Command Completed with Success
5	SCB Command Completed with Success after Retries
7	Subsystem Hardware Failure
A	Immediate Command Completed
C	Command Completed with Failure
E	Command Error (Invalid Command or Parameter)
F	Software Sequencing Error

Note: All values not shown are reserved.

Figure 14. Interrupt ID Table

Bits 3 – 0 These bits identify the device reporting the interrupt.

Hex Value	Device Selected	SCSI Physical Unit Number	SCSI Logical Unit Number
0	Logical Device 0	0	0
1	Logical Device 1	1	0
2	Logical Device 2	2	0
3	Logical Device 3	3	0
4	Logical Device 4	4	0
5	Logical Device 5	5	0
6	Logical Device 6	6	0
7	Logical Device 7*	Not Assigned	Not Assigned
8	Logical Device 8	Not Assigned	Not Assigned
9	Logical Device 9	Not Assigned	Not Assigned
A	Logical Device 10	Not Assigned	Not Assigned
B	Logical Device 11	Not Assigned	Not Assigned
C	Logical Device 12	Not Assigned	Not Assigned
D	Logical Device 13	Not Assigned	Not Assigned
E	Logical Device 14	Not Assigned	Not Assigned
F	Subsystem	**	**

* The device number that is equal to the SCSI ID of the subsystem is not assigned after Reset. The default value for this SCSI ID is 7.

** Logical Device 15 is reserved for the subsystem.

Note: These assignments may be changed by POST.

Figure 15. Subsystem Device Selection Defaults

Basic Status Register (Hex 07)

This 8-bit read-only register returns subsystem status information to the system.

Bit	Symbol
7−4	Reserved
3	Command Interface Register Full
2	Command Interface Register Empty
1	Interrupt Request
0	Busy

Figure 16. Basic Status Register

Bits 7−4 These bits are reserved.

Bit 3 This bit is set to 1 after all Command Interface registers are loaded. It is set to 0 when any one of these registers is read by the subsystem.

Bit 2 This bit is set to 1 after all Command Interface registers are read by the subsystem. It is set to 0 by the subsystem when any one of the Command Interface registers is loaded.

Bit 1 This bit is set to 1 when the subsystem is presenting an interrupt to the system. This bit is set to 0 when an EOI request code is written to the Attention register. This bit will be set to 1 if an interrupt request is active, even if the Basic Control register interrupt enable bit is set to 0. This allows the subsystem to operate without presenting hardware interrupts to the system.

Bit 0 This bit is set to 1 when the Attention register is loaded or when a hardware reset occurs. It is set to 0 by the subsystem after the Attention register and Command Interface registers have been read or the hardware reset is completed. If a Reset command is issued to the subsystem, this bit is set to 0 after completing the reset function and before setting the command complete interrupt.

Resetting the Subsystem

When the subsystem is reset, diagnostic routines are run to determine if the subsystem is functioning properly. The subsystem can be reset by:

- CHRESET on the system channel going active.
- Setting bit 7 (hardware reset) in the Basic Control register to 1.
- Issuing the Immediate Reset command to the subsystem (device 15). This command performs a soft reset of the subsystem.

A soft reset occurs when the Reset command is issued to a SCSI device or to the subsystem (device 15). When the command is issued to the subsystem, the subsystem activates the 'reset' signal on the SCSI bus, which resets all devices on the bus and clears all pending commands. When the command is issued to any other SCSI device, only that device is reset.

A hard reset occurs when the system 'reset' signal (+CHRESET) is pulsed active or when bit 7 in the Basic Control register is set to 1. A hard reset does not reset the SCSI bus or any attached devices; it does, however, reset the subsystem.

After the reset sequence is completed, results are posted in the Interrupt Status register. After a hardware reset, bits 7–4 are set as shown in the following figure; bits 3–0 are always set to hex F.

Note: After a soft reset, hex AF indicates a successful reset; hex 7F indicates a hardware failure.

Hex	Interrupt
0	No Error - Reset Completed
1	Microprocessor ROM Test Failed
2	Local RAM Test Failed
3	Power Protection Device Error
4	Microprocessor Internal Peripheral Test Failed
5	Buffer Control Chip Test Failed
6	Reserved
7	System Interface Control Chip Test Failed
8	SCSI Interface Test Failed

Note: All values not shown are reserved.

Figure 17. Subsystem Reset Status

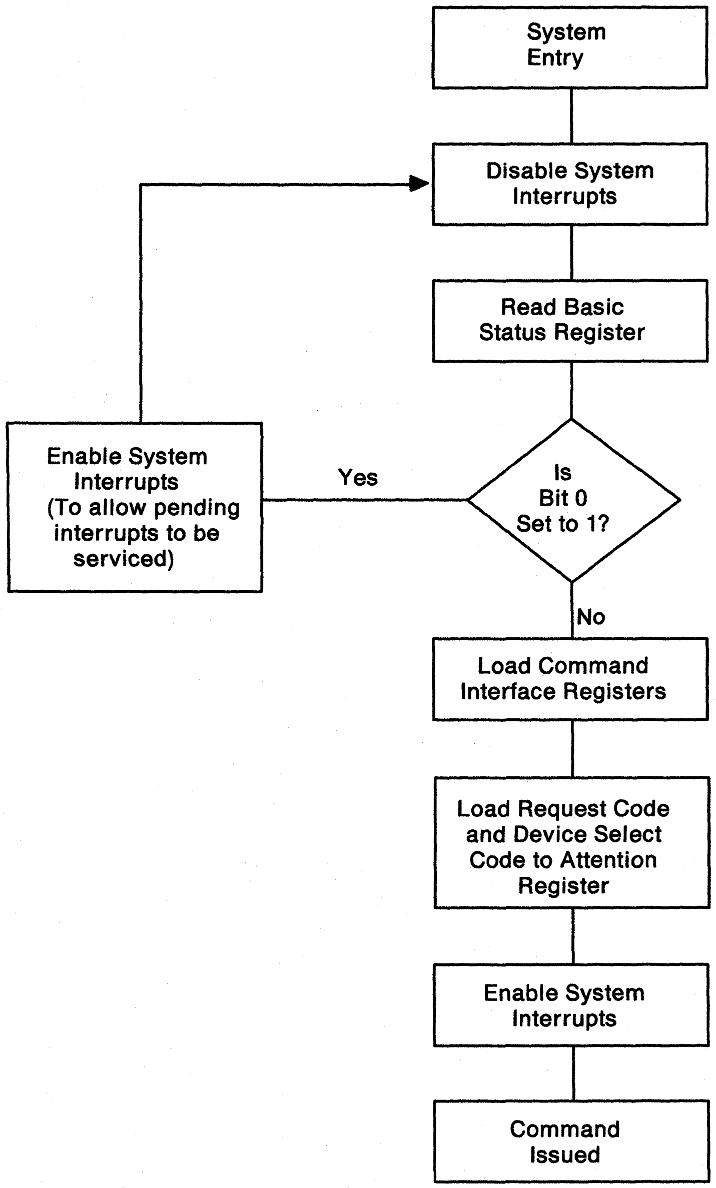
If a failure occurs during the reset sequence, the subsystem will accept only a hard reset. This prevents accidental loss of data.

Command Processing

Four 8-bit command interface registers are used in conjunction with the Attention register to issue commands to the subsystem. The data written to the Command Interface registers consists of either a 32-bit immediate command or a 24-bit address that points to a subsystem control block stored in system memory. The data written to the Attention register specifies the device-select code and a request code. The request code indicates if the Command Interface registers contain an immediate command or a pointer to a subsystem control block. See “System Interface” on page 25 for more information about supported commands.

One command can be specified for each of 15 SCSI logical devices and one global subsystem command, for a maximum of 16 overlapped commands in progress. New commands can be issued even while a data transfer is in progress for a previous command. If a command is directed to a device with a command already in progress, the command in progress is ended with a command sequence error, and the new command is ignored. Before starting a command, make sure bit 0 in the Basic Status register is set to 0, and the system interrupts are disabled.

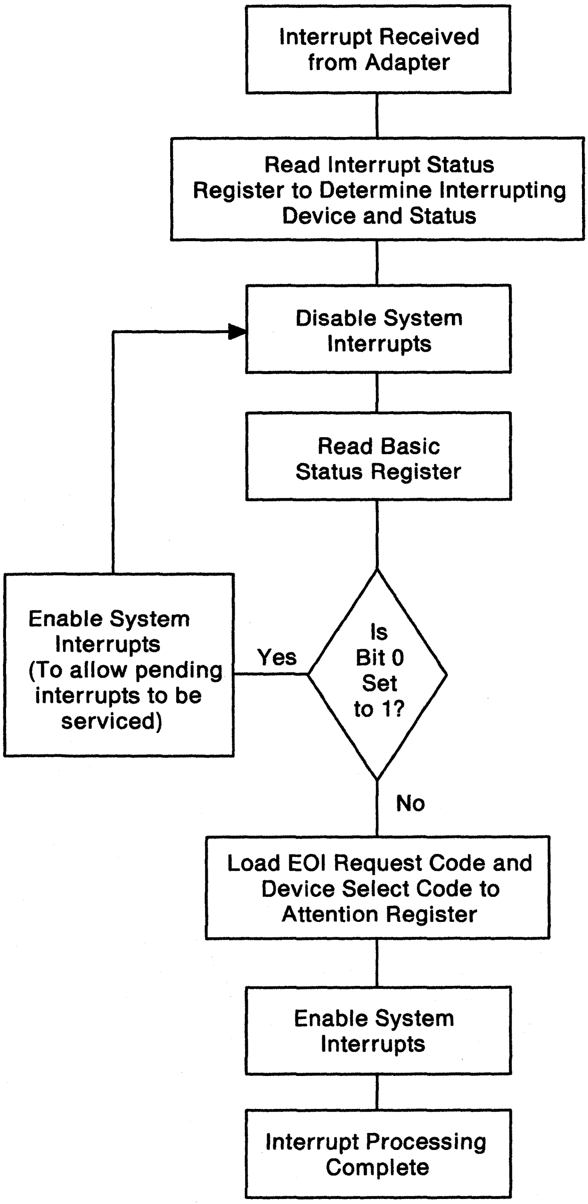
The following flowchart shows the sequence for issuing a command.



System Entry
Disable System Interrupts
Read Basic Status Register
Is Bit 0 Set to 1?
Yes
Enable System Interrupts (To allow pending interrupts to be serviced)
No
Load Command Interface Registers
Load Request Code and Device Select Code to Attention Register
Enable System Interrupts
Command Issued

Figure 18. Issuing a Command

The following flowchart shows the sequence for processing an interrupt.



Interrupt Received from Adapter
Read Interrupt Status Register to Determine Interrupting Device and Status
Disable System Interrupts
Read Basic Status Register
Is Bit 0 Set to 1?
Yes
Enable System Interrupts (To allow pending interrupts to be serviced)
No
Load EOI Request Code and Device Select Code to Attention Register
Enable System Interrupts
Interrupt Processing Complete

Figure 19. Processing an Interrupt

Note: If the busy bit has not cleared within 400 microseconds, the application performs time-out procedures. When a subsystem Reset command is in progress, allow up to 30 seconds for the busy bit to clear.

After a command is completed, the subsystem interrupts the system by loading the Interrupt Status register with summary status (the completing device number and an interrupt ID). The subsystem waits for the EOI request before it issues another interrupt or changes the contents of the Interrupt Status register.

When the system reads the Interrupt Status register, the interrupting device number and status may be obtained. The Command Complete status block contains more status information about the completed command.

Note: To get the status block, use the Get Command Complete Status command, or inspect the termination status block, if available.

The system then loads the interrupting device number and the EOI request into the Attention register. This tells the subsystem that the system has finished processing that interrupt.

Subsystem Control Blocks

The Subsystem Control Block commands relieve the system of transferring command blocks to the subsystem through programmed input and output. The SCB specifies the desired command and associated parameters. An SCB must begin on a doubleword boundary, and any address translations, from virtual to physical, must be performed by system software before the SCB pointer is loaded into the Command Interface registers. If 80386 virtual page mode is being used, system software must also ensure that the SCB, data buffers, and termination status block (TSB) areas are locked into memory. The SCB specifies the desired command and associated parameters. When the SCB command (or chain of SCB commands) has been performed by the subsystem, an interrupt request is issued to the system. The subsystem presents only one interrupt request at a time to the system.

The following figure shows the format of the subsystem control block as it applies to device-related commands.

```
15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0 Remarks

<---- Command Dependent ---> ND NS C5 C4 C3 C2 C1 C0 Command Word
RD  ES  RE  PT   0  SS  BB 0  0  0  0  0  0  0  0 CH Enable Word
<------------ Least Significant Word --------------> Logical Block
<------------ Most Significant Word --------------->  Address
<------------ Least Significant Word --------------> System-Buffer
<------------ Most Significant Word --------------->  Address
<------------ Least Significant Word --------------> System-Buffer
<------------ Most Significant Word --------------->  Byte Count
<------------ Least Significant Word --------------> Termination-Status-Block
<------------ Most Significant Word --------------->  Address
<------------ Least Significant Word --------------> Optional SCB-Chain
<------------ Most Significant Word --------------->  Address
<------------ Number of Blocks --------------------> Block Count
<------------ Block Size --------------------------> Block Length
```

Figure 20. Subsystem Control Block Structure

A command is encoded in the first word of the SCB. The setting of bits 15 – 8 of the first word depends on the specific command identified in bits 5 – 0. If bit 7 (ND) of this word is set to 1, the subsystem will not disconnect the target device during command execution. If bit 6 (NS) of this word is set to 1, the subsystem will not send any Synchronous Data Transfer Request messages to the target device.

The second word of the SCB is used to enable options that are used to modify a specified command, as shown in the following figure.

Bit	Symbol	Function
15	RD	Input/Output Control
14	ES	Report TSB Status Only on Error
13	RE	Retry Enable
12	PT	Pointer to List
10	SS	Suppress Exception Short
9	BB	Bypass Buffer
8 – 1		Reserved
0	CH	Chain on No Error

Figure 21. Enable Options

Bit 15 (RD)	When this bit is set to 1, the subsystem transfers data from the SCSI device or subsystem into system memory (read). When this bit is set to 0, the subsystem transfers data from system memory to the SCSI device or subsystem (write).
Bit 14 (ES)	When this bit is set to 1, the TSB is transferred to memory only if an error (Interrupt ID = C) is detected. When this bit is set to 0, the TSB is always transferred. **Note:** This bit should always be set to 1, unless the command requires the TSB when no error occurs; command performance is degraded by unnecessarily writing to memory.
Bit 13 (RE)	This bit is included for reference only. This bit is not used nor checked by the subsystem.
Bit 12 (PT)	When this bit is set to 1, it allows a single command to write data to or read data from several different areas in memory (buffers) as specified in a list. This list contains up to 16 pairs of values, each pair being a 32-bit address and its related 32-bit count. In the SCB, the system-buffer address field contains the address of the list, and the system-buffer byte count field contains the length of the list in bytes.
Bit 10 (SS)	When this bit is set to 1, it allows the amount of data transferred on a read operation to be shorter than the system buffer byte count, specified in the SCB, without generating an error.
Bit 9 (BB)	This bit is included for reference only. This bit is not used nor checked by the subsystem.
Bits 8 – 1	These bits are reserved.
Bit 0 (CH)	This bit selects the type of chaining condition used in command block transfers. When it is set to 0, chaining is disabled. When command blocks are chained, the SCB must contain the 32-bit address of the next SCB. When this bit is set to 1, chaining will occur if the SCB ends with no error.

System Interface

The following is a list of supported commands.

Command Type	Command	Hex Code	Supported Device Numbers (Hex)	Page Reference
Immediate	Abort	0F	0−F	26
Immediate	Assign	0E	F	27
SCB	Device Inquiry	0B	0−E	28
Immediate	DMA Pacing Control	0D	F	29
Immediate	Feature Control	0C	0−F	30
Immediate	Format Prepare	17	0−E	31
SCB	Format Unit	16	0−E	31
SCB	Get Command Complete Status	07	0−F	34
SCB	Get POS Information	0A	F	40
SCB	Read Data	01	0−E	43
SCB	Read Device Capacity	09	0−E	44
SCB	Read Verify	03	0−E	45
SCB	Reassign Block	18	0−E	46
SCB	Request Sense	08	0−E	47
Immediate	Reset	00	0−F	51
SCB	Send Other SCSI	1F	0−E	52
SCB	Write Data	02	0−E	53
SCB	Write with Verify	04	0−E	54

Figure 22. Subsystem Command List

Note: The hex code above represents bits 5−0 of the first command word.

The subsystem maintains a Command Complete status block for each of the command blocks. The command blocks are updated at the completion of each command. This command status block can be obtained by using the Get Command Complete Status Block command. See “Command Complete Status Block” on page 34.

The format for each command is given following the associated command.

Abort

This immediate command causes the logical device to immediately end the command in progress and go to the bus-free state. This command sends an Identify message followed by an Abort message to the device. An Abort command to the subsystem clears an active global command.

After the command is completed, the system can request the status block of the cancelled command. The status block shows the state of the operation at the time it was ended.

The Abort command is issued only when the system has timed out while waiting for the subsystem to complete a command. In response to this command, the subsystem stops the current command and maintains command status information. This information may be used to determine where the command problem occurred. The subsystem interrupts the system when it has completed the Abort command.

```
              Command Interface Register Bits
  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0     Remarks

   0   0   0   0   0   1  0  0  0  0  0  0  1  1  1  1     Abort
  <------------------- Reserved ---------------------->
```

Figure 23. Abort Command

Assign

This immediate command assigns a device number (0 to 14) to a particular SCSI device. This allows any 15 SCSI devices to have command processing active at one time. The Assign command must be directed to the subsystem (device 15).

When the SCSI device is assigned a device number, subsystem retries are enabled for that device number.

Notes:

1. A device cannot be assigned the same SCSI physical unit number (PUN) as the subsystem SCSI ID set in POS Register 3.
2. A hardware reset sets all device number assignments to the default value. See Figure 15 on page 16.
3. Device number assignments may be removed (devices 0 to 14 not assigned) by issuing the Assign command, with bit 7 (R) of the second word of the command block set to 1.
4. If the device number being assigned to a SCSI logical unit (LUN) has a command in progress when the Assign command is received, the attachment will end the command with an error, and the assignment will not be changed.
5. Only one device number can be assigned to a particular SCSI device. If an attempt is made to assign more than one device number to the same SCSI device, the command will end with an error (Interrupt ID = C).

```
          Command Interface Register Bits
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0     Remarks

  0  0  0  0  0  1  0  0  0  0  0  1  1  1  0        Assign
 <--- Reserved ---> <LUN>   R <PUN>   <Dev>          Logical Unit,
                                                     Physical Unit,
                                                     and Device No.
```

Figure 24. Assign Command

Device Inquiry

Through the SCB Device Inquiry command, the system determines which SCSI devices are attached to the subsystem and specific information about those devices. When the Device Inquiry data block has been transferred, the subsystem interrupts the system. Because the length of the returned data block is device dependent, the system should specify the amount of data to be returned. If this is not known, then the system should specify the maximum value (255) and set the suppress short exception (SS) bit to 1. After the Device Inquiry data block is transferred to the specific address, the subsystem interrupts the system to indicate that the command is complete.

If a SCSI device is not attached at the assigned physical SCSI address, the command-completed-with-failure interrupt will be returned in the Interrupt Status register. The command complete status will indicate selection time-out. If the SCSI logical unit number is not supported by an attached SCSI physical unit, the device type in the Device Inquiry data block is set to hex 7F by the SCSI physical device.

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0    Remarks

 0  0  0  1  1  1  0  0 ND NS  0  0  1  0  1  1    Device Inquiry
 1 ES RE  0  0 SS  1  0  0  0  0  0  0  0  0 CH    Enable Word
<---------- Reserved ----------------------->
<---------- Reserved ----------------------->
<---------- Least Significant Word ---------->    System-Buffer
<---------- Most Significant Word ----------->     Address
<---------- Least Significant Word ---------->    System-Buffer
<---------- Most Significant Word ----------->     Byte Count
<---------- Least Significant Word ---------->    Termination-Status-Block
<---------- Most Significant Word ----------->     Address
<---------- Least Significant Word ---------->    Optional SCB-Chain
<---------- Most Significant Word ----------->     Address
<---------- Reserved ----------------------->
<---------- Reserved ----------------------->
```

Figure 25. Device Inquiry Command

Device Inquiry Data Block

```
Byte   7 6 5 4 3 2 1 0    Remarks

 0     <Peripheral Device Type>   Major Type
 1     RMB <- Type Qualifier ->   Removable Media Bit
 2     <ISO>  <-ECMA-> <-ANSI->   Standards Compliance
 3     <------ Reserved ------>
 4     <- Additional Length -->   No. Of Bytes (N-4)
 - - - - - - - - - - - - - - - - - - - - - - - - - - -
 5     <-- Additional Data --->   Additional

                 .                Inquiry

 N     <-- Additional Data --->   Data


 ECMA - European Computer Manufacturer's Association
 ISO  - International Standards Organization
```

Figure 26. Device Inquiry Data Block

For more information about the Device Inquiry data block, refer to the American National Standards Institute SCSI Standard X3.131-1986.

DMA Pacing Control

This immediate command is issued to the subsystem (device 15). It controls the pacing of DMA transfers by the subsystem. The pacing factor is specified as a percentage of the total bandwidth the subsystem is allowed to use. The acceptable range is 25 to 100 percent.

The pacing value is used until a new value is specified or until a subsystem power-on or Micro Channel* reset occurs. A power-on or Micro Channel reset will set pacing to 100% (no pacing).

```
          Command Interface Register Bits
15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0     Remarks

 0   0   0   0   0   1  0  0  0  0  0  0  1  1  0  1     Pacing
 <------ Reserved ---------> <---Pacing Factor -->       Control
```

Figure 27. DMA Pacing Control Command

* Micro Channel is a trademark of the International Business Machines Corporation.

Feature Control

This immediate command controls various features of the SCSI subsystem. The command time-out limit (in seconds) can be specified through bits 12–0 of the second word of the command block. A value of 0 prevents the subsystem from timing out. The time-out for a command defaults to 45 seconds upon a subsystem hardware reset. When this command is issued to the subsystem (device 15), the maximum SCSI bus synchronous-data-transfer rate allowed by the subsystem can be specified through bits 15–13 of the second word of the command block. The following figure describes the relationship between the setting of bits 15–13 and the maximum data-transfer rate.

Bits 15 14 13	Rate Specified (Millions of Bytes per Second)
0 0 0	5.00 (power-on default)
0 0 1	4.00
0 1 0	3.33
0 1 1	2.86
1 0 0	2.50
1 0 1	2.22
1 1 0	2.00
1 1 1	1.82

Figure 28. Device Data-Transfer Rate

When this command is issued to the subsystem (device 15), the data rate and time-out value applies to all devices. When this command is issued to a device (device 0–14), the data-rate control bits are ignored and the time-out value applies only to the device to which the command was issued.

```
           Command Interface Register Bits
 15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0     Remarks

  0   0   0   0   0   1  0  0  0  0  0  0  1  1  0  0     Feature
  Max SCSI                                                Control
  < D-Rate > <---- Command Global Time-out Value ---->    Data-Transfer
                                                           Rate
```

Figure 29. Feature Control Command

Format Prepare

This immediate command acts as a format interlock to prevent inadvertent data destruction. The Format Prepare command must be issued immediately prior to the Format Unit command. If another command is issued between the Format Prepare command and the Format Unit command, the format is not performed and ends in an error.

Command Interface Register Bits																
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Remarks**
0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	1	Format Prepare
0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0	55AA

Figure 30. Format Prepare Command

Format Unit

This SCB command is used to format a storage device. Formatting the storage device destroys all data. The device performs defect management as specified in the command. Bits within the command specify the source of the defect list and the use and disposition of any defect list on the device.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	ND	NS	0	1	0	1	1	0	Format Unit
0	ES	RE	0	0	0	1	0	0	0	0	0	0	0	0	CH	Enable Word
<------ Reserved ---->								0	0	0	FD	CL	0	0	0	Modifier Bits
<---------- Interleave Factor ---------------->																Interleave
<---------- Least Significant Word ----------->																System-Buffer
<---------- Most Significant Word ------------>																Address
<---------- Least Significant Word ----------->																Defect-List
<---------- Most Significant Word ------------>																Byte Count
<---------- Least Significant Word ----------->																Termination-Status-Block
<---------- Most Significant Word ------------>																Address
<---------- Least Significant Word ----------->																Optional SCB-Chain
<---------- Most Significant Word ------------>																Address
<---------- Number of Blocks ----------------->																Block Count
<---------- Block Size ----------------------->																Block Length

Figure 31. Format Unit Command

The interleave factor used during the format operation is specified in the control block. An interleave factor of 0 selects the device default. A factor of 1 selects sequential numbering of logical blocks. All other factor values are device dependent.

Modifier bits select options to be used during formatting and are defined as follows:

FD **Format Data:** When this modifier bit is set to 1, the system supplies a defect list for the format operation. The structure of the list depends on the device being formatted. The system-buffer address points to the defect list; the length is specified in the byte count. If this bit is set to 0, no defect list is transferred to the device.

Note: Not all SCSI devices support the transfer of a defect list.

CL **Complete List:** If the defect list is supplied, this bit determines whether the supplied defect list is in addition to, or replaces, the defect list already in the device. If the bit is set to 1, any previous defect list is replaced.

Note: Only a defect list in the following block format is supported by the subsystem. See the ANSI SCSI Standard or specific device specification for more information.

```
            Defect List Header
Byte       7  6  5  4  3  2  1  0    Remarks


 0         <----- Reserved ----->
 1         <----- Reserved -BF-->
 2         <----- High Byte ---->
 3         <----- Low Byte ----->
             Defect Descriptors
 4         <----- High Byte ---->    First
 5         <-------------------->    Defective Block
 6         <-------------------->    Address
 7         <----- Low Byte ----->
                      .
                      .
                      .
           <----- High Byte ---->    Last
           <-------------------->    Defective Block
           <-------------------->    Address
 N         <----- Low Byte ----->
```

Figure 32. Defect List Block Format

BF **Background Format:** When this bit is set to 1, the device performs a background format. If the device supports this option, it checks the format of the command, then returns a command status indicating good status, and starts the format operation. If the device does not support the option, it may return a command status block indicating a check condition.

Commands received before completing the background format are returned with a command status block indicating a check condition. The request sense command returns a sense key indicating that the device is not ready and returns an additional sense code indicating that a Format operation is in progress. The request sense data block also shows the percentage of the format completed.

Get Command Complete Status

This SCB command requests the command complete status block for the last command executed on a specified device. When the status block is transferred to the system, the subsystem generates an interrupt and updates the Interrupt Status register.

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0   Remarks

 0  0  0  1  1  1  0  0  0  0  0  0  1  1  1      Get Command Status
 1 ES RE  0  0  0  1  0  0  0  0  0  0  0 CH      Enable Word
 <---------- Reserved ----------------------->
 <---------- Reserved ----------------------->
 <---------- Least Significant Word ---------->   System-Buffer
 <---------- Most Significant Word ----------->    Address
 0  0  0  0  0  0  0  0  0  0  0  1  1  0  1  0   System-Buffer
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0    Byte Count
 <---------- Least Significant Word ---------->   Termination-Status-Block
 <---------- Most Significant Word ----------->    Address
 <---------- Least Significant Word ---------->   Optional SCB-Chain
 <---------- Most Significant Word ----------->    Address
 <---------- Reserved ----------------------->
 <---------- Reserved ----------------------->
```

Figure 33. Get Command Complete Status Command

Command Complete Status Block

The command complete status block is returned to the location specified in the system-buffer address field of the Get Command Complete Status command. It contains the status of the last command to a device. It is unchanged until another command is issued to that device or until a reset occurs.

An optional termination status block is returned automatically whenever an error (Interrupt ID = C) occurs. This allows command complete status to be returned for error recovery. See Figure 21 on page 23 for more information.

The command complete status block and termination status block contain the same information.

Note: A Get Command Complete Status command returns valid status information following a hardware error interrupt. For errors (Interrupt ID = 7, E, and F) caused by hardware problems *do not* use the termination status block; the subsystem internally cancels a command at the point of the hardware error.

```
Word    15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0     Remarks

  0     <---------- SCB End Status Word -------------->     SCB Status
  1     <---------- Reserved-------------------------->
  2     <---------- Least Significant Word ----------->     Residual Byte
  3     <---------- Most Significant Word ------------>     Count
  4     <---------- Least Significant Word------------>     Scatter/Gather
                                                             List
  5     <---------- Most Significant Word ------------>     Element Address
  6     0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0      Device-Dependent
                                                             Status Length
  7     <-- Command Status --><---- Device Status ---->     Command Device
                                                             Status
  8     <-- Command Error ---><---- Device Error ----->     Error Codes
  9     <---------- Reserved ------------------------->
  A     <---------- Reserved ------------------------->
  B     <---------- Least Significant Word ----------->     Last SCB Address
  C     <---------- Most Significant Word ------------>      Processed
```

Figure 34. Command Complete Status Block

Word 0 - Subsystem Control Block End Status

Bit	Function
15 – 13	Reserved
12	Major Exception Occurred
11	Device Not Initialized
10	Reserved
9	Device-Dependent Status Available
8	Additional Status Available
7	SCB Interrupt Queued
6	SCB Halted (Error/End Chain)
5	Long Record Exception
4	SCB Specification Check
3	SCB Rejected
2	Invalid Command Rejected
1	Short Record Exception
0	SCB Ended (No Error)

Note: The function indicated is true when the value of the bit is one. Reserved bits are undefined.

Figure 35. SCB End Status

Word 1 - Reserved

Words 2 and 3 - Residual Byte Count: These words contain the number of bytes that were not transferred.

Words 4 and 5 - Scatter/Gather List Element Address: These words contain the address of the scatter/gather list element being used when the command was ended.

Word 6 - Device-Dependent Status Length: This word contains the number of bytes of device status information that follow. This word is set to hex 0C to indicate 12 bytes.

Word 7 - Command and Device Status

Hex	Command Status
1	SCB Command Completed with Success
5	SCB Command Completed with Success after Retries
7	Subsystem Hardware Failure
A	Immediate Command Completed
C	Command Completed with Failure
E	Command Error (Invalid Command or Parameter)
F	Software Sequencing Error

Note: All values not shown are reserved.

Figure 36. Command Status Codes

Bit	Function
7	Reserved
6	Vendor Unique Bit
5	Vendor Unique Bit
4−1	Device Status Code
0	Vendor Unique Bit

Figure 37. Device Status Byte

Hex	Device Status
0	Good Status (No Error)
1	Check Condition (Error)
2	Condition Met/Good (No Error)
4	Busy (Error)
8	Intermediate/Good (No Error)
A	Intermediate/Condition Met/Good (No Error)
C	Reservation Conflict (Error)

Note: All values not shown are reserved.

Figure 38. Bits 4−1 Device Status Code

Word 8 - Command Error Code/Device Error Code

Hex	Error
00	No Error
01	Invalid Parameter in SCB
02	Reserved
03	Command Not Supported
04	Command Ended (By System)
05	Reserved
06	Reserved
07	Format Rejected - Sequence Error
08	Assign Rejected - Command in Progress on Device
09	Assign Rejected - SCSI Device Already Assigned
0A	Command Rejected - SCSI Device Not Assigned
0B	Maximum Logical Block Address Exceeded
0C	16-Bit Card Slot Address Range Exceeded
0D – 12	Reserved
13	Invalid Device for Command
14 – 1F	Reserved
20	Subsystem Hardware Error
21	Global Command Time-out
22	DMA Error
23	Subsystem Buffer Defective
24	Command Ended by Subsystem
25 – 7F	Reserved
80	Subsystem Microprocessor Detected Error
81 – FF	Reserved

Figure 39. Bits 15 – 8 Command Error Code

Hex	Error
00	No Error
01	SCSI Bus Reset Occurred
02	SCSI Interface Fault
03 – 0F	Reserved
10	SCSI Selection Time-out (device not available)
11	Unexpected SCSI Bus Free
12	Reserved
13	Invalid SCSI Phase Sequence
14 – 1F	Reserved
20	Short Length Record
21 – FF	Reserved

Figure 40. Bits 7 – 0 Device Error Code

Word 9 - Reserved

Word A - Reserved

Word B - Last SCB Address Processed - Low Word

Word C - Last SCB Address Processed - High Word

Get POS and Subsystem Information

This SCB command requests the subsystem to return POS information and subsystem parameters.

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0    Remarks

 0  0  0  1  1  1  0  0  0  0  0  0  1  0  1  0    Get POS Information
 1 ES RE  0  0  0  1  0  0  0  0  0  0  0  0 CH    Enable Word
 <---------- Reserved ----------------------->
 <---------- Reserved ----------------------->
 <---------- Least Significant Word ---------->    System-Buffer
 <---------- Most Significant Word ----------->     Address
 0  0  0  0  0  0  0  0  0  0  1  0  0  1  0       System-Buffer
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0    Byte Count
 <---------- Least Significant Word ---------->    Termination-Status-Block
 <---------- Most Significant Word ----------->     Address
 <---------- Least Significant Word ---------->    Optional SCB-Chain
 <---------- Most Significant Word ----------->     Address
 <---------- Reserved ----------------------->
 <---------- Reserved ----------------------->
```

Figure 41. Get POS and Subsystem Information Command

POS and Subsystem Information Status Block

```
Word  15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0  Remarks

 0     1  0  0  0  1  1  1  0  1  1  1  1  1  1  1  0  Subsystem ID
 1     <--- POS Register 2 ---><-- POS Register 3 -->  POS Regs 2-3
 2     <--- POS Register 4 ---><-- Interrupt Level-->  POS Reg 4/Intr Level
 3     <- Size --><---- Revision Level ------------->  Subsystem Level
 4     <-No. Devices Supported-><No. LUNs per Device>  No. Devices/
                                                        No. LUNs per Device
 5     <- No. Logical Dev Nos.-><-- Pacing Factor -->  No. of Logical
                                                        Dev Nos. Pacing
                                                        Factor
 6     <--- Max Busy Time ---><-- EOI to Intro Off ->  Max Busy Time/
                                                        EOI to Int off
 7     <--------------- Reserved ------------------->
 8     <--------------- Reserved ------------------->
```

Figure 42. POS and Subsystem Information Status Block

Word 0 - Subsystem POS ID = Hex 8EFE

Bits 15 – 8	These bits contain hex 8E.
Bits 7 – 0	These bits contain hex FE.

Word 1 - POS Registers 2 and 3

Bits 15 – 8	These bits contain the current value for POS Register 2.
Bits 7 – 0	These bits contain the current value for POS Register 3.

Word 2 - POS Register 4/Subsystem Interrupt Level

Bits 15 – 8	These bits contain the current value for POS Register 4.
Bits 7 – 0	These bits contain Interrupt level, hex 0E.

Word 3 - Channel Connector Size/Subsystem Revision Level

Bits 15 – 12 These bits indicate if the subsystem is currently installed in a 16- or 32-bit Micro Channel connector.

Bits 15 14 13 12	**Channel Connector**
0 0 0 0	32-Bit
0 0 0 1	16-Bit

Figure 43. Channel Connector Size

Bits 11 – 0 These bits are reserved.

Word 4 - Number of Devices Supported/Number of Logical Units per SCSI Device

Bits 15 – 8	These bits contain the number of physical units supported (up to 7).
Bits 7 – 0	These bits contain the number of logical units supported (up to 8).

Word 5 - Number of Logical Device Numbers Supported/Pacing Factor

Bits 15 – 8	These bits contain the number of logical devices supported (16).
Bits 7 – 0	These bits contain the DMA Pacing factor (%) as set by the Immediate Pacing control command.

Word 6 - Maximum Subsystem Busy Time/EOI to Interrupt Off Time

Bits 15 – 8	These bits specify the time (up to 30 seconds) from hardware reset to busy off.
Bits 7 – 0	These bits specify the time (one microsecond) from load of the End of Interrupt request code until the 'hardware interrupt' signal is deactivated.

Word 7 - Reserved

Word 8 - Reserved

Read Data

This SCB command is used for devices with fixed-length blocks, such as fixed disk drives. This command causes the subsystem to send the SCSI Read command to the device. The blocks specified are read and the data is transferred to the system.

The Read Data command supports multiple block operations up to 65 535 blocks or 16MB minus 1 byte (MB equals 1 048 576 bytes), whichever is less, of total data transferred.

For devices with variable length blocks, such as tape drives, the Send Other SCSI SCB command should be used to generate the SCSI Read command.

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0    Remarks

 0  0  0  1  1  1  0  0 ND NS  0  0  0  0  0  1    Read Data
 1 ES RE PT  0  0 BB  0  0  0  0  0  0  0  0 CH    Enable Word
 <---------- Least Significant Word ---------->    Logical
 <---------- Most Significant Word ----------->     Address
 <---------- Least Significant Word ---------->    System-Buffer
 <---------- Most Significant Word ----------->     Address
 <---------- Least Significant Word ---------->    System-Buffer
 <---------- Most Significant Word ----------->     Byte Count
 <---------- Least Significant Word ---------->    Termination-Status-Block
 <---------- Most Significant Word ----------->     Address
 <---------- Least Significant Word ---------->    Optional SCB-Chain
 <---------- Most Significant Word ----------->     Address
 <---------- Number of Blocks ---------------->    Block Count
 <---------- Block Size ---------------------->    Block Length
```

Figure 44. Read Data Command

Read Device Capacity

This SCB command is used to return the Device Capacity status block of the specific device.

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0    Remarks

 0  0  0  1  1  1  0  0 ND NS  0  0  1  0  0  1    Read Device Capacity
 1 ES RE  0  0  0  1  0  0  0  0  0  0  0  0 CH    Enable Word
 <---------- Reserved ----------------------->
 <---------- Reserved ----------------------->
 <---------- Least Significant Word ---------->    System-Buffer
 <---------- Most Significant Word ----------->     Address
 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0    System-Buffer
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0     Byte Count
 <---------- Least Significant Word ---------->    Termination-Status-Block
 <---------- Most Significant Word ----------->     Address
 <---------- Least Significant Word ---------->    Optional SCB-Chain
 <---------- Most Significant Word ----------->     Address
 <---------- Reserved ----------------------->
 <---------- Reserved ----------------------->
```

Figure 45. Read Device Capacity Command

Device Capacity Data Block

```
Byte       7  6  5  4  3  2  1  0      Remarks

 0         <----- High Byte ------>
 1         <----------------------->     Last Logical
 2         <----------------------->     Block Address
 3         <----- Low Byte ------->
 4         <----- High Byte ------>
 5         <----------------------->     Block
 6         <----------------------->     Length
 7         <----- Low Byte ------->
```

Figure 46. Device Capacity Data Block

Read Verify

This SCB command reads the specified blocks of data and checks for errors. Data is not transferred by this command; it serves to verify the readability of the data and the correct operation of the device. This command is used for devices with fixed-length blocks, such as fixed disk drives. This command causes the subsystem to send the SCSI Read and Verify commands to the device. The blocks specified are read and the data is transferred to the system.

The Read Verify command supports multiple block operations up to 65 535 blocks or 16MB minus 1 byte (MB equals 1 048 576 bytes), whichever is less, of total data transferred.

For devices with variable length blocks, such as tape drives, the Send Other SCSI SCB command should be used to generate the SCSI Read and Verify commands.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	ND	NS	0	0	0	0	1	1	Read Verify
1	ES	RE	0	0	0	1	0	0	0	0	0	0	0	0	CH	Enable Word
<---------- Least Significant Word ---------->																Logical Block
<---------- Most Significant Word ----------->																Address
<---------- Reserved ----------------------->																
<---------- Reserved ----------------------->																
<---------- Reserved ----------------------->																
<---------- Reserved ----------------------->																
<---------- Least Significant Word ---------->																Termination-Status-Block
<---------- Most Significant Word ----------->																Address
<---------- Least Significant Word ---------->																Optional SCB-Chain
<---------- Most Significant Word ----------->																Address
<---------- Number of Blocks ---------------->																Block Count
<---------- Block Size --------------------->																Block Length

Figure 47. Read Verify Command

Reassign Block

This SCB command reassigns the logical block address for a defective block to a spare block. The system supplies the reassign block defect list. The system-buffer address in the command block serves as a pointer to the defect list.

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0    Remarks

 0  0  0  1  1  1  0  0  0  0  0  1  1  0  0  0    Reassign Block
 0 ES RE  0  0  0  1  0  0  0  0  0  0  0  0 CH    Enable Word
<---------- Reserved ------------------------>
<---------- Reserved ------------------------>
<---------- Least Significant Word ----------->    System-Buffer
<---------- Most Significant Word ------------>     Address
<---------- Least Significant Word ----------->    System-Buffer
<---------- Most Significant Word ------------>     Byte Count
<---------- Least Significant Word ----------->    Termination-Status-Block
<-----------Most Significant Word ------------>     Address
<---------- Least Significant Word ----------->    Optional SCB-Chain
<---------- Most Significant Word ------------>     Address
<---------- Reserved ------------------------>
<---------- Reserved ------------------------>
```

Figure 48. Reassign Block Command

Reassign Block Defect List

```
               Defect List Header
Byte          7  6  5  4  3  2  1  0    Remarks

 0            <----- Reserved ----->
 1            <----- Reserved ----->
 2            <----- High Byte ---->    Defect List
 3            <----- Low Byte ----->    Length
                Defect Descriptors
 4            <----- High Byte ---->
 5            <-------------------->    Defective Logical
 6            <-------------------->    Block
 7            <----- Low Byte ----->    Address
```

Figure 49. Reassign Block Defect List

Request Sense

This SCB command is used to return the sense data for the specified device. The subsystem interrupts the system when the Sense data block is transferred. The length of the data block depends on the device and can be four bytes (non-extended) or more (extended). The format of the data block for both cases is shown. The system should specify the amount of data to be returned in the SCB based on the particular device attached, or specify the maximum value (255) and set the suppress short exception (SS) bit to 1.

The sense data is valid only if a Check Condition status was returned for the previous command to the device. The sense data provides additional information on the check condition. Refer to the ANSI SCSI publication or the particular SCSI device specification for detailed information about the request sense data block.

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0    Remarks

 0  0  0  1  1  1  0  0 ND NS  0  0  1  0  0  0    Request Sense
 1 ES RE  0  0 SS  1  0  0  0  0  0  0  0  0 CH    Enable Word
<---------- Reserved ----------------------->
<---------- Reserved ----------------------->
<---------- Least Significant Word ---------->    System-Buffer
<---------- Most Significant Word ----------->     Address
<---------- Least Significant Word ---------->    System-Buffer
<---------- Most Significant Word ----------->     Byte Count
<---------- Least Significant Word ---------->    Termination-Status-Block
<---------- Most Significant Word ----------->     Address
<---------- Least Significant Word ---------->    Optional SCB-Chain
<---------- Most Significant Word ----------->     Address
<---------- Reserved ----------------------->
<---------- Reserved ----------------------->
```

Figure 50. Request Sense

Sense Data Block

```
Byte              Sense Bits
           7  6  5  4  3  2  1  0    Remarks

 0         AV <Class>  <- Code ->    Error Class/Code
 1         X  X  X  < High Byte >    Logical
 2         <------------------->     Block
 3         <----- Low Byte ----->    Address
```

Figure 51. Sense Data Block

Byte 0 Error Class/Error Code

Bit 7 **Address Valid:** When this bit is set to 1, the logical block address field is valid.

Bits 6 – 4 **Error Class:** When the error class is 0, the sense data block is in the format shown above. When the error class is 7, the sense data block is in the extended format, shown on the following page. All other settings are device dependent.

Bits 3 – 0 **Error Code:** Errors are device dependent.

Bytes 1 – 3 **Logical Block Address:** This address is device dependent.

Note: The subsystem does not examine or use device-dependent information.

Extended Sense Data Block

```
Byte                Sense Bits
          7  6  5  4  3  2  1  0    Remarks

 0        V  1  1  1 <-- Code -->   Error Class/Code
 1        <---- Segment No. ---->   Segment Number
 2        FM EM IL  X <-- Key -->   Sense Key
 3        <- Most Significant -->   Information Bytes
 4        <--------------------->
 5        <--------------------->
 6        <- Least Significant ->
 7        <--Additional Length-->   No. of Bytes
 8        <--Additional Sense -->   Additional
                    .               Sense
                    .
 N        <--Additional Sense--->   Bytes
```

Figure 52. Extended Sense Data Block

Byte 0 Error Class/Error Code

Bit 7 The Information bytes are valid only if this bit is a 1.

Bits 6 – 4 Error class 7 is for extended-sense data.

Bits 3 – 0 Error code 0 is standard format. Error codes hex 1 - E are reserved. Error code hex F is device dependent.

Byte 1 Segment Number: This byte contains the current segment descriptor.

Byte 2 Extended Error Bits/Sense Key

Bit 7 A filemark (FM) has been reached on a sequential access device.

Bit 6 An end of medium (EM) has been reached on a sequential access device.

Bit 5 An Invalid Length (IL) resulted when the specified logical block length did not match the device.

Bit 4 X - This bit is reserved.

Bits 3 – 0 The coding of these bits is shown in the following figure.

Hex Value	Function
0	No Sense
1	Recovered Error
2	Not Ready
3	Medium Error
4	Hardware Error
5	Illegal Request
6	Unit Attention
7	Data Protect
8	Blank Check
9	Device Dependent
A	Copy Ended
B	Command Ended
C	Equal
D	Volume Overflow
E	Miscompare
F	Reserved

Figure 53. Sense Key

Bytes 3 – N Device-Dependent Status: Refer to the particular device specifications for a definition of these bytes.

Note: The subsystem does not examine or use device-dependent information.

Reset

This immediate command enables the system to reset a specific physical device or globally reset the subsystem and all attached devices. A Reset command issued to a SCSI device causes the subsystem to send a SCSI Bus-Device Reset message to the corresponding physical device. The SCSI Bus-Device Reset message causes the physical device to immediately end all commands in progress on all logical units attached, and go to the bus-free state. The system can reset the subsystem by a soft reset if, for example, the system times out while waiting to complete an Abort command.

To reset the subsystem, issue a Reset command to device hex F. The subsystem stops all current activity and all attached devices are reset by activating the SCSI 'reset' signal. When the Reset command is completed, the subsystem sets the Interrupt Status register to indicate the results. When the Command Complete interrupt occurs, the system reads the Interrupt Status register and checks for the Immediate Command Complete interrupt (hex A). A value of hex 7F in the Interrupt Status register indicates a diagnostic routine within the subsystem has detected an error and the subsystem may be defective. Individual devices can be reset as required for initialization or error recovery.

```
          Command Interface Register Bits
 15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0     Remarks

  0   0   0   0   0   1  0  0  0  0  0  0  0  0  0  0     Reset
  <----------------- Reserved -------------------->
```

Figure 54. Reset Command

Send Other SCSI Command

This SCB command is used to send any SCSI command not supported by the subsystem directly to a SCSI device. The command to be issued is placed at the end of the SCB. When commands are issued directly to a device using this command, messages are handled by the subsystem. Data transfer direction is controlled by the read-option bit (RD) in the Enable word. When this bit is set to 1, the subsystem transfers data to the system from the device. When the read-option bit is set to 0, the subsystem transfers data to the device from the system. If the system-buffer byte count specified in the SCB is 0, no data is transferred.

Notes:

1. This command should be used only when other commands cannot perform the operation; otherwise, performance of the SCSI subsystem can be impacted.
2. This command should be issued only to logical device numbers 0 to 14.

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0    Remarks

 0  0  1  0  0  1  0  0 ND NS  0  1  1  1  1  1    Send Other SCSI Command
RD ES RE PT  0 SS  1  0  0  0  0  0  0  0  0 CH    Enable Word
<------ Reserved -----><-- SCSI CMD Length --->
<---------- Reserved ------------------------->
<---------- Least Significant Word ----------->    System-Buffer
<---------- Most Significant Word ------------>     Address
<---------- Least Significant Word ----------->    System-Buffer
<---------- Most Significant Word ------------>     Byte Count
<---------- Least Significant Word ----------->    Termination-Status-Block
<---------- Most Significant Word ------------>     Address
<---------- Least Significant Word ----------->    Optional SCB-Chain
<---------- Most Significant Word ------------>     Address
<-------1------- SCSI Command -------0-------->    SCSI Command
<-------3------- SCSI Command -------2-------->
<-------5------- SCSI Command -------4-------->    Six Bytes or
<-------7------- SCSI Command -------6-------->
<-------9------- SCSI Command -------8-------->    Ten Bytes or
<------11------- SCSI Command ------10-------->    Twelve Bytes
```

Figure 55. Send Other SCSI Command

Write Data

This SCB command writes data from the system to the device in consecutive blocks. This command is used for devices with fixed-length blocks, such as fixed disk drives. This command causes the subsystem to send the SCSI Write command to the device. The blocks specified are read and the data is transferred to the system. No verification is performed.

The Read Data command supports multiple block operations up to 65 535 blocks or 16MB minus 1 byte (MB = 1 048 576 bytes), whichever is less, of total data transferred.

For devices with variable length blocks (such as tape drives) the Send Other SCSI SCB command should be used to generate the SCSI Write command.

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0    Remarks

 0  0  0  1  1  1  0  0 ND NS  0  0  0  0  1  0    Write Data
 0 ES RE PT  0  0 BB  0  0  0  0  0  0  0  0 CH    Enable Word
 <---------- Least Significant Word ----------->   Logical-Block
 <---------- Most Significant Word ------------>    Address
 <---------- Least Significant Word ----------->   System-Buffer
 <---------- Most Significant Word ------------>    Address
 <---------- Least Significant Word ----------->   System-Buffer
 <---------- Most Significant Word ------------>    Byte Count
 <---------- Least Significant Word ----------->   Termination-Status-Block
 <---------- Most Significant Word ------------>    Address
 <---------- Least Significant Word ----------->   Optional SCB-Chain
 <---------- Most Significant Word ------------>    Address
 <---------- Number of Blocks ----------------->   Block Count
 <---------- Block Size ----------------------->   Block Length
```

Figure 56. Write Data Command

Write with Verify

This SCB command is similar to Write Data, except that a Read Verify command is performed after all blocks are written. This command is used for devices with fixed length blocks, such as fixed disk drives. This command causes the subsystem to send the SCSI Write and Verify commands to the device. The blocks specified are read and the data is transferred to the system.

The Write with Verify command supports multiple block operations up to 65 535 blocks or 16MB minus 1 byte (MB = 1 048 576 bytes), whichever is less, of total data transferred.

If an error occurs during a Write with Verify command, the system should retry the command. If all retries of the command fail, the system can allocate a spare block to replace the failing one through the Reassign Block command, and then reissue the command.

For devices with variable length blocks, such as tape drives, the Send Other SCSI SCB command should be used to generate the SCSI Write and Verify commands.

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0    Remarks

 0  0  0  1  1  1  0  0 ND NS  0  0  0  1  0  0    Write Verify
 0 ES RE PT  0  0 BB  0  0  0  0  0  0  0  0 CH    Enable Word
 <---------- Least Significant Word ----------->    Logical-Block
 <---------- Most Significant Word ------------>     Address
 <---------- Least Significant Word ----------->    System-Buffer
 <---------- Most Significant Word ------------>     Address
 <---------- Least Significant Word ----------->    System-Buffer
 <---------- Most Significant Word ------------>     Byte Count
 <---------- Least Significant Word ----------->    Termination-Status-Block
 <---------- Most Significant Word ------------>     Address
 <---------- Least Significant Word ----------->    Optional SCB-Chain
 <---------- Most Significant Word ------------>     Address
 <---------- Number of Blocks ----------------->    Block Count
 <---------- Block Size ----------------------->    Block Length
```

Figure 57. Write with Verify Command

Error-Recovery Procedures

This section describes the error-recovery procedures for the subsystem and the system.

Subsystem-Error Recovery

The subsystem performs the error-recovery procedures (ERP) as necessary to ensure data integrity and maintain system availability. The subsystem will not perform retries regardless of the setting of RE command option bit in the SCB.

Device level retries are device dependent and are usually disabled through the SCSI Mode Select command.

Data Error

The following section will discuss read operations, write operations, and format operations in data-error recovery.

Read Operations: SCSI devices automatically perform read-error recovery procedures unless this function is disabled.

When a device error persists, the logical block causing the error can be reassigned to a spare block by using the Reassign Block command. The subsystem does not reassign a defective block on its own.

Write Operations: Each device performs its own write-error recovery procedures unless this function is disabled.

If a device error persists, the logical block in error can be reassigned. The subsystem does not reassign defective blocks on its own.

Format Operations: Each device performs its own format-error recovery procedures unless retries have been disabled. If a device returns a format error, and all retries are unsuccessful, the subsystem terminates the command with an error.

Positioning Error

Each device performs its own positioning-error-recovery procedures unless error recovery is disabled. If a device returns an error indicating a positioning error, and all retries were unsuccessful, the subsystem ends the command with an error.

SCSI-Bus Parity Error

If a SCSI-bus parity error occurs, the subsystem retries the failing sequence unless retries have been disabled. If the retry is unsuccessful, or if the total number of errors reaches nine, the command is terminated immediately with a nonrecoverable error.

Operational Time-Outs

The subsystem ensures timely completion of commands through local and global time-outs. If a SCSI device does not respond to a selection sequence within the time-out period, a selection time-out error will be returned to the system. If a command is not completed within the time-out period, a global time-out error is returned to the subsystem.

The subsystem time-out values are 260 milliseconds for a selection time-out, and 45 seconds (default), which is changed through the Set Features command, for a global time-out.

System-Error Recovery

The system must process errors reported after a reset or command completion. Some types of errors are normal. For instance, a unit attention error after a device is reset, or after removable media is changed, is normal. When an error is reported that is not expected or not normal, the system should follow the error-recovery and fault-isolation procedure table. This procedure helps determine if the error is recoverable, and if it is the result of a command error, a system programming error, or a subsystem hardware failure. Recommended actions are indicated for each class of command, based on the interrupt ID returned.

The following two figures describe error-recovery procedures (ERP).

Command Class	**Interrupt ID**	**Recommended Action**
Device Reset Command	AX	ERP Complete/Retry Original Commands
	7X	ERP 19
	CX	ERP 18
	EX	ERP 11
	FX	ERP 14
	Others	ERP 18
	Time-out	ERP 19
Device Abort Command	AX	ERP Complete/Retry Original Commands
	7X	ERP 19
	CX	ERP 16
	EX	ERP 11
	FX	ERP 14
	Others	ERP 18
	Time-out	ERP 18
Request Sense Commands	1X	ERP 3
	5X *	ERP 9
	7X	ERP 19
	CX	ERP 15
	EX	ERP 11
	FX	ERP 14
	Others	ERP 18
	Time-out	ERP 18
All Other Device Commands	1X	No Errors
	5X	ERP 8
	7X	ERP 19
	CX	ERP 1
	EX	ERP 11
	FX	ERP 14
	Others	ERP 18
	Time-out	ERP 18

Figure 58. SCSI Command ERP Table

Command Class	Interrupt ID	Recommended Action
Subsystem Reset Command	AF	ERP Complete/Retry Original Commands
	7F	ERP 20
	CF	ERP 18
	EF	ERP 11
	FF	ERP 14
	Others	ERP 19
	Time-out	ERP 19
Subsystem Abort Command	AF	ERP Complete/Retry Original Commands
	7F	ERP 20
	CF	ERP 19
	EF	ERP 11
	FF	ERP 14
	Others	ERP 19
	Time-out	ERP 19
All Other Subsystem Commands	1F	No Errors
	5F	ERP 8
	7F	ERP 20
	EF	ERP 11
	FF	ERP 14
	Others	ERP 17
	Time-out	ERP 17
Hardware Reset	0F	ERP Complete/Retry Original Commands
	3F	ERP 20
	8F	ERP 21
	Others	ERP 22
	Time-out	ERP 22

Figure 59. Subsystem Command ERP Table

The following error-recovery procedures assume that subsystem and device retries have not been disabled.

ERP 1 If the termination status block or command complete status block indicates a Check condition in the device status byte, go to ERP 2. Otherwise, go to ERP 12.

ERP 2 Issue the Request Sense command to the SCSI device. See the appropriate ERP table.

ERP 3 If the Sense key from the request-sense data equals 6 (Unit Attention), go to ERP 6. Otherwise, go to ERP 4.

ERP 4 If the Sense key from the request-sense data equals 2 (Not Ready), go to ERP 5. Otherwise, go to ERP 15.

ERP 5	If the removable media bit (RMB) is set to 1 (bit 7 in the Device Inquiry data block), the program should handle media installation (for example, prompt for media insertion), then go to ERP 7. If the RMB is set to 0, go to ERP 15.
ERP 6	If the RMB is set to 1, the program should take any action needed when the media is changed, then go to ERP 7. If the RMB is set to 0, go to ERP 7.
ERP 7	Retry the original operation three times or more. If an error persists, go to ERP 15.
ERP 8	Log the soft error counts from TSB or command-complete status block (CCSB) if desired. Continue normal operations.
ERP 9	Log the soft error counts from TSB or CCSB if desired. Go to ERP 3.
ERP 10	Retry the original operation at least three times. If an error persists, go to ERP 17.
ERP 11	Check the command and the parameters for validity. Check the TSB or CCSB command error code for additional information.
ERP 12	If the TSB or CCSB command error code is hex OA (device not assigned), suspect a system software initialization or configuration problem. If none is found, proceed to ERP 13.
ERP 13	If the TSB or CCSB device error code equals hex 10 (Selection Time-Out), make sure the SCSI device is properly connected to the SCSI bus. Also, be sure that the SCSI address (ID) on the device is set properly and is not the same as any other device (including the subsystem SCSI ID set in POS register 3). Otherwise, proceed to ERP 18.
ERP 14	Check the system software command queuing/delivery code for a possible problem. If none is found, go to ERP 18.
ERP 15	Issue an Immediate Abort command to the SCSI device. See the appropriate ERP table.
ERP 16	Issue an Immediate Reset command to the SCSI device. See the appropriate ERP table.
ERP 17	Issue an Immediate Abort command to the subsystem. See the appropriate ERP table.

ERP 18 Issue an Immediate Reset command to the subsystem. See the appropriate ERP table.

ERP 19 Perform a hardware reset through the Basic Control register bit 7. Set bit 7 to 1. Then, reset bit 7 to 0 to perform the reset. See the appropriate ERP table.

ERP 20 Verify the proper connection of the SCSI bus cables and termination networks. If an error persists, proceed to ERP 21. Otherwise, replace the defective SCSI bus cable or the defective termination network.

ERP 21 Remove the devices from the SCSI bus, one at a time, and perform a hardware reset until the error condition is removed. If all the devices and cables (except the terminators) are removed and an error persists, proceed to ERP 22. Otherwise, replace the defective SCSI device, the defective SCSI bus cable, or the defective termination network. See the appropriate ERP table.

ERP 22 Replace the defective subsystem. Restore the system to the original configuration. See the appropriate ERP table.

Compatibility

The subsystem uses the SCB architecture as an interface to the system. Some differences, however, exist between the architecture definition and how the subsystem implements the architecture. This section describes those differences.

Commands

The subsystem does not support all commands defined by the architecture. In addition, the subsystem does not support a command hierarchy. The architecture allows the Suspend and the Reset Interrupt Status commands to be accepted by a device while it is performing another command, and the subsystem does not.

The following SCB commands defined by the architecture are not supported by the subsystem:

- Suspend
- Resume
- Initialize Device
- Non-operation.

The Non-operation Immediate command defined by the architecture is not supported by the subsystem.

Rejected Commands

The architecture defines that a subsystem indicate that a command has been rejected by setting the reject bit in a status area called the Command Status byte. Under the following conditions, the subsystem generates a command-sequence interrupt:

- The device interrupt queue is full
- The attention code is not valid
- The Immediate-command operation code is not valid
- The device is busy
- The device is not available
- The device specified is not valid.

Command Options

The architecture allows optional facilities to be selected for commands sent to each device; these options are selected using Enable word (Word 1) of the control block. The Enable word also allows for subsystems, such as the subsystem, to define certain bits for their specific needs.

The subsystem does not use some of these options; bits selecting these options are reserved on the subsystem and must be set to 0.

The following shows the differences in the options between the subsystem and the architecture.

- SCB Interrupts on Command Complete
  - **Architecture:** Allows an optional interrupt on any SCB. The architecture will allow any SCB in a chain to request an SCB interrupt.
  - **Subsystem:** Issues an interrupt for every SCB not in a chain. The subsystem will not allow an interrupt for normal SCB completion except for the last SCB in the chain.
- SCB Interrupts on Non-SCB Command Complete
  - **Architecture:** Allows optional specification on some commands. Disallows interrupts on other commands to avoid recursion problems with full interrupt queues.
  - **Subsystem:** Issues an interrupt after every command.
- Storing End Status Word 1 on an SCB Interrupt for Normal Completion
  - **Architecture:** Compatible with the subsystem when the Device Interrupt Identification register is not supported.
  - **Subsystem:** Stores TSB word 0 only when requested by SCB Enable word.
- Storing of Residual Buffer Address in the TSB on Errors
  - **Architecture:** To standardize error handling, always store a Buffer address in the TSB whenever a Residual Byte count is provided.
  - **Subsystem:** Stores buffer address only when Indirect list is used in SCB.

- Suppress Exception Long in SCB Enable Word 1
  - **Architecture:** This is an optional feature.
  - **Subsystem:** The subsystem does not support this option.
- Extended SCBs and TSBs
  - **Architecture:** This is an optional feature that allows usage of two operand-storage addresses including incrementing and decrementing.
  - **Subsystem:** The subsystem does not support this option.

Interrupts

The following section describes the differences between the SCB architecture definition of interrupts and the subsystem implementation of the architecture.

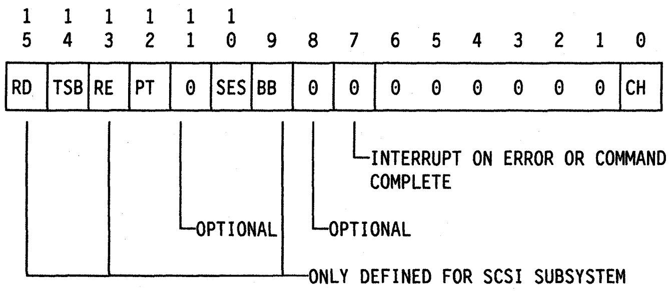
- Presentation of Interrupt Status in the Interrupt Status Register
  - **Architecture:** Supports two modes of Interrupt handling. In the simple mode, all interrupts flow through the Interrupt Status register, which is compatible with the subsystem implementation. In the Device Interrupt Identification register mode, SCB interrupts are not presented through the Interrupt Status register.
  - **Subsystem:** All interrupts flow through a single Interrupt Status register.
- Command to Clear the Interrupt Status Register
  - **Architecture**: Compatible with subsystem implementation when the Device Identification Interrupt register extension to the architecture is not implemented.
  - **Subsystem**: Implements only a single command, and can clear only a single interrupt at a time.
- Clearing Multiple SCB Logical Interrupts on a Single Command
  - **Architecture**: Supported by Reset SCB interrupt command when the Device Identification Interrupt register is supported.
  - **Subsystem**: This is not supported.
- Presentation of Multiple Causes on Physical Interrupt
  - **Architecture**: Multiple causes presented in the Device Identification Interrupt register method of interrupt handling.
  - **Subsystem**: Only one cause per Physical interrupt.
- Interrupt Code Returned for Reset Device
  - **Architecture**: Returns Interrupt code hex 0 (reset complete) to allow a program to discard interrupt data in the Interrupt Status port for a device when a Reset Device is issued.
  - **Subsystem**: Returns Interrupt code hex A (immediate command complete) no error.

Other Differences

SCB Enable Word 1

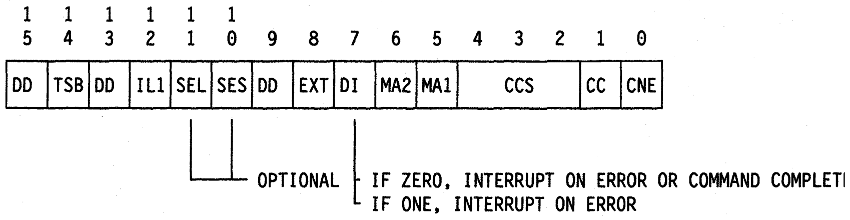
This section describes the differences between the SCB architecture definition of the SCB Enable Word 1 and how the subsystem implements the architecture.

Subsystem



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
RD TSB RE PT 0 SES BB 0 0 0 0 0 0 0 0 CH
INTERRUPT ON ERROR OR COMMAND COMPLETE
OPTIONAL
OPTIONAL
ONLY DEFINED FOR SCSI SUBSYSTEM

SCB Architecture



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DD TSB DD IL1 SEL SES DD EXT DI MA2 MA1 CCS CC CNE
OPTIONAL
IF ZERO, INTERRUPT ON ERROR OR COMMAND COMPLETI
IF ONE, INTERRUPT ON ERROR

Figure 60. SCB Enable Word 1

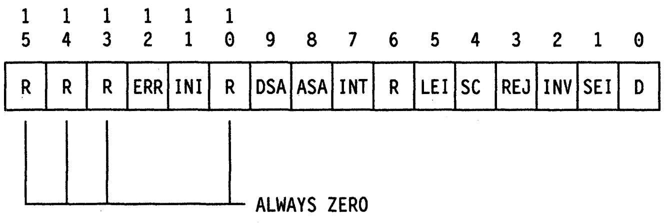
Note: Consider the following when examining both Enable Words.

- Bits shown as 0 in the subsystem are reserved and must be set to 0. These bits are upward compatible with the architecture.
- Bits CH and CNE are the same (chain no error).
- Bits PT and IL1 are the same (indirect list specified).
- Bits marked as DD are usable for subsystem-defined functions. This is compatible with the architecture.

SCB End Status Word 1

This section describes the differences between the SCB architecture definition of the SCB End Status Word 1 and how the subsystem implements the architecture.

Subsystem



SCB Architecture

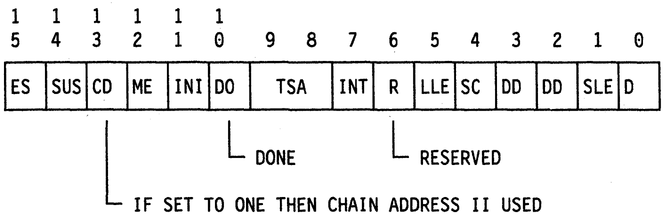

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ES SUS CD ME INI DO TSA INT R LLE SC DD DD SLE D
DONE
RESERVED
IF SET TO ONE THEN CHAIN ADDRESS II USED

Figure 61. SCB End Status Word 1

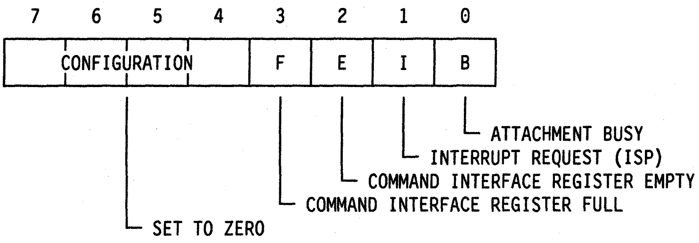
Note: Consider the following when examining the TSB End Status Word 1 and the SCSI TSB Word Zero.

- Bits shown as R for the subsystem are reserved and must be set to 0. These bits are upward compatible with the architecture.
- Bits SEI and SLE are the same (short length exception detected.)
- Bit TSA is upward compatible with bits ASA and DSA. These bits indicate the amount of TSB status stored.
- Bits ME and ERR are the same. These bits indicate that a major error or exception has occurred.
- Bits marked as DD in the architecture are device dependent.

SCB Command Status Register

This section describes the differences between the SCB architecture definition of the SCB Command Status register and how the subsystem implements the architecture.

Subsystem



SCB Architecture

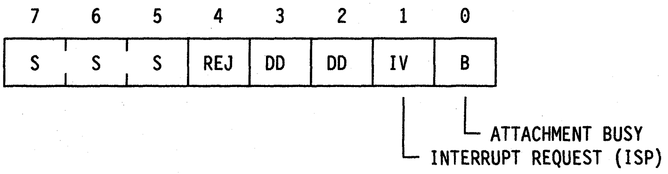


Figure 62. SCB Command Status Register

Note: Consider the following when examining the Command Status registers.

- Bits I and IV are the same. Both show that an interrupt value has been placed in the Interrupt Status register.
- Bits marked as DD are usable for subsystem defined functions.

SCB Subsystem Control Register

This section describes the differences between the SCB architecture definition of the SCB Subsystem Control register and how the SCSI subsystem implements the architecture.

Subsystem

```
  7     6     5     4     3     2     1     0
+-----+-----+-----+-----+-----+-----+-----+-----+
|  R  |  0  |  0  |  0  |  0  |  0  |  D  |  I  |
+-----+-----+-----+-----+-----+-----+-----+-----+
   |     |                                   |
   |     |                                   └─ ENABLE INTERRUPTS
   |     └─ SETUP
   |
   └─ RESET SUBSYSTEM
```

SCB Architecture

```
  7     6     5     4     3     2     1     0
+-----+-----+-----+-----+-----+-----+-----+-----+
| RST |  R  | RR  |  R  | SD  | SD  | DMA | EI  |
+-----+-----+-----+-----+-----+-----+-----+-----+
   |           |                       |     |
   |           |                       |     └─ ENABLE INTERRUPTS
   |           |                       └─ ENABLE DMA
   |           └─ RESET REJECT
   └─ RESET SUBSYSTEM
```

Figure 63. SCB Subsystem Control Register

Note: The following points should be considered when examining the Basic Control register and the Subsystem Control register.

- Bits shown as 0 in the subsystem are reserved and must be set to 0. These bits are upward compatible with the architecture.
- Bits I and EI are the same. These bits enable the subsystem to send interrupts to the system unit.
- Bits D and DMA are the same. These bits enable the subsystem to continue to use DMA services.
- Bits R and RST are the same. These bits perform a hardware controlled reset of the subsystem.
- Bits marked as SD are usable for subsystem defined functions. This is compatible with the architecture.

SCB Immediate Command Format

This section describes the differences between the SCB architecture definition of the SCB Immediate command format and how the subsystem implements the architecture.

Subsystem

```
COMMAND INTERFACE REGISTER FORMAT 0

 3                                1   1   1   1
 1                                6   5   4   3 ◄──── 9 8 7◄──────── 0
┌──────────────────────────────────┬───┬───┬─────────────┬─┬───────────────┐
│        OP CODE DEPENDENT         │ 0 │ 0 │ OP CODE DEP │0│ IMMEDIATE CMD │
└──────────────────────────────────┴───┴───┴─────────────┴─┴───────────────┘
```

SCB Architecture

```
COMMAND INTERFACE REGISTER FORMAT 0

 31                               16  15  14  13 ◄──── 9 8 7◄──────── 0
┌──────────────────────────────────┬───┬───┬─────────────┬─┬───────────────┐
│        OP CODE DEPENDENT         │DCI│DDI│ OP CODE DEP │0│ IMMEDIATE CMD │
└──────────────────────────────────┴───┴───┴─────────────┴─┴───────────────┘

COMMAND INTERFACE REGISTER FORMAT 1

 31                               16  15  14  13 ◄──── 9 8 7◄──────── 0
┌──────────────────────────────────┬───┬───┬─────────────┬─┬───────────────┐
│             RESERVED             │DCI│DDI│ OP CODE DEP │1│ IMMEDIATE CMD │
└──────────────────────────────────┴───┴───┴─────────────┴─┴───────────────┘
```

Figure 64. SCB Immediate Command Format

Note: The following points should be considered when examining immediate command format zero:

- The Immediate CMD codes assigned are compatible.
- The DDI bit is compatible with subsystem implementation. When set to 0, this bit leaves the device enabled for interrupts. When set to one, this bit disables the device from sending interrupts to the system unit.

- The DCI bit is compatible with subsystem implementation. When set to zero, this bit requests an interrupt on completion. When set to one, this bit does not request an interrupt on completion of an immediate command. The architecture is a superset when this bit is not ignored by the command (Reset Interrupt Status register.)

This section describes additional differences between the SCB architecture definition and how the subsystem implements the architecture.

- Specification Testing of SCBs
  - **Architecture**: This is an optional feature.
  - **Subsystem**: Compatible with architecture.
- Device Number for Subsystem
  - **Architecture**: Subsystem is device hex 00.
  - **Subsystem**: Subsystem is device hex 0F.
- Attention Code to Start Long SCB
  - **Architecture**: These are device-dependent attention codes in the architecture. They are compatible with the subsystem implementation.
  - **Subsystem**: Uses hex 04 or hex 0F.
- Attention Code Hex 00.
  - **Architecture**: Uses as device reset.
  - **Subsystem**: Not implemented. Does a device reset under Attention Code hex 01.

- Interrupt Status Codes are the same except for:
  - **Architecture**: Uses

    0 - Reset device or subsystem complete
    2 - Notify event when Locate mode commands use Move mode command delivery
    5 - Device dependent (compatible with implementation)
    6 - Inform event when Locate mode commands use Move mode command
    8 - Hardware failure reading control block command or storing TSB status.
    E - Command rejected when Locate mode commands use Move mode command delivery
    D - Non-SCB command error
    F - Device dependent
  - **Subsystem**: Uses

    0 - Reset subsystem complete
    2 - Reserved
    5 - SCB Completed with retries
    6 - Reserved
    8 - Reserved
    E - Invalid command
    D - Reserved
    F - Software sequence error
- Disabling Device on Immediate Commands
  - **Architecture**: Controlled by the DDI in the Immediate command format.
  - **Subsystem**: Enables devices by default. The subsystem does not support this option. This is a proper subset of the architecture.

Device Signals

The subsystem supports single-ended drivers and receivers. All signal lines are digital, open collector transistor-transistor logic (TTL), and provide signals to the various SCSI devices. The drivers have the following electrical specifications:

Active: 0.0 V dc to 0.5 V dc at 48 milliampere (maximum)

Inactive: 2.5 V dc to 5.25 V dc at 250 microampere (open collector)

The following are the signal descriptions for the subsystem-to-device connector.

-Acknowledge (-ACK)	This signal is driven by the subsystem to acknowledge the request for data transfer.
-Attention (-ATN)	This signal is driven by the subsystem to initiate the transfer of a message to a device.
-Busy (-BSY)	This signal is the OR input from all devices on the bus, including the subsystem. It is driven active by a device to indicate that the bus is in use.
-Control/Data (-C/D)	This signal indicates if the data bus contains control or data information. When the signal is low, the data bus contains control information. This signal is driven by the device.
-Data Bits (-D0-7,DP)	These signals contain the eight data bits plus the one parity bit that make up the data bus. Data bit 7 is the most-significant bit and has the highest priority during arbitration. The subsystem uses odd parity for all transfers except arbitration. When the signal is low, the respective bit is a logical 1.

-Input/Output (-I/O)	This signal indicates the direction of the signal flow regarding the subsystem. When this signal is low, the data is either being sent to the subsystem, or the 'select' signal is being driven by a device. This signal is driven by the device.
-Message (-MSG)	This signal is driven by a device to indicate that the data bus contains a message.
-Request (-REQ)	This signal is driven by a device to initiate a data transfer.
-Reset (-RST)	This signal is driven by the subsystem to reset all SCSI devices. This will occur as a result of a subsystem Reset command.
-Select (-SEL)	This signal is bidirectional and indicates that the data bus contains a device ID. This signal is either driven by the subsystem to select a device, or driven by the device to reselect the subsystem.

Connectors

The subsystem has two connectors. The internal connector is a keyed 2- by 25-pin header located on the right side of the system board. This connector supports SCSI devices mounted inside the system unit. Even-numbered pins are on the main component side of the subsystem, with pin 2 at the end closest to the D-shell connector.

The following figure shows the signals and pin assignments for the internal connector.

Pin	Signal	Pin	Signal
1	Ground	2	-Data Bits(0)
3	Ground	4	-Data Bits(1)
5	Ground	6	-Data Bits(2)
7	Ground	8	-Data Bits(3)
9	Ground	10	-Data Bits(4)
11	Ground	12	-Data Bits(5)
13	Ground	14	-Data Bits(6)
15	Ground	16	-Data Bits(7)
17	Ground	18	-Data Bits(P)
19	Ground	20	Ground
21	Ground	22	Ground
23	Ground	24	Ground
25	Not Connected	26	+Terminator Power
27	Ground	28	Ground
29	Ground	30	Ground
31	Ground	32	-Attention
33	Ground	34	Ground
35	Ground	36	-Busy
37	Ground	38	-Acknowledge
39	Ground	40	-Reset
41	Ground	42	-Message
43	Ground	44	-Select
45	Ground	46	-Control/Data
47	Ground	48	-Request
49	Ground	50	-Input/Output

Figure 65. SCSI Internal Connector Pin Assignments

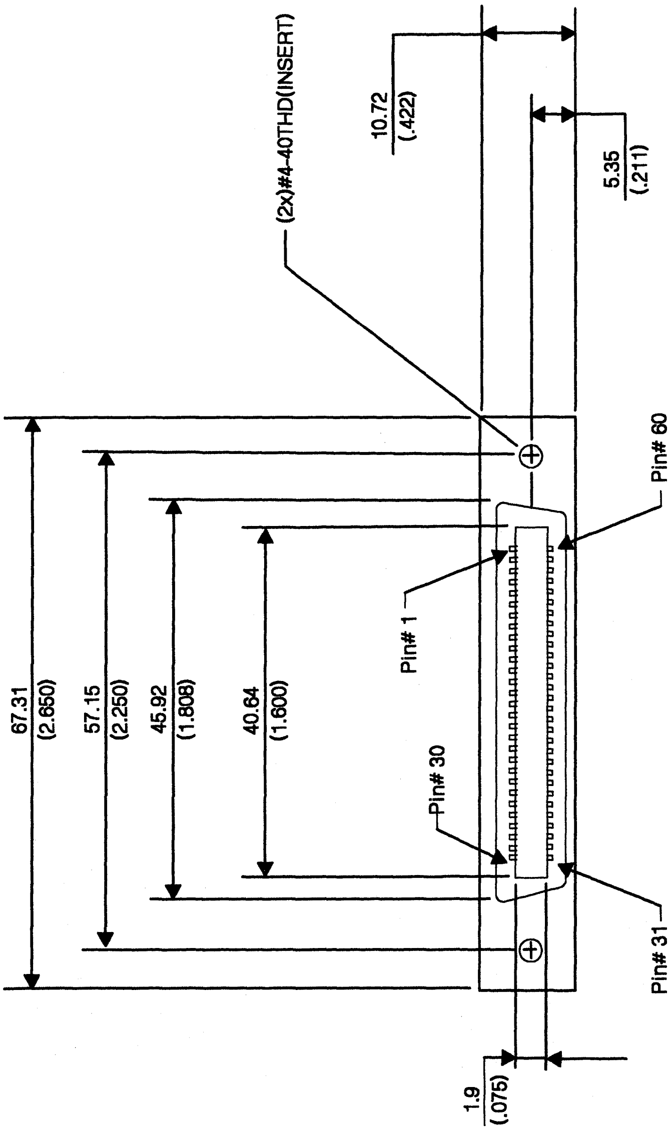
A 60-pin external connector on the rear of the subsystem allows attachment of external SCSI devices. Pins 1 through 50 correspond to pins 1 through 50 of the SCSI Standard. Pin 51 can be used as ground.

The following figure shows the signals and pin assignments for the external connector. See Figure 67 on page 77 for pin location on the external connector.

Pin	Signal	Pin	Signal
1	Ground	31	Ground
2	-Data Bits(0)	32	-Attention
3	Ground	33	Ground
4	-Data Bits(1)	34	Ground
5	Ground	35	Ground
6	-Data Bits(2)	36	-Busy
7	Ground	37	Ground
8	-Data Bits(3)	38	-Acknowledge
9	Ground	39	Ground
10	-Data Bits(4)	40	-Reset
11	Ground	41	Ground
12	-Data Bits(5)	42	-Message
13	Ground	43	Ground
14	-Data Bits(6)	44	-Select
15	Ground	45	Ground
16	-Data Bits(7)	46	-Control/Data
17	Ground	47	Ground
18	-Data Bits(P)	48	-Request
19	Ground	49	Ground
20	Ground	50	-Input/Output
21	Ground	51	Ground
22	Ground	52	Reserved
23	Ground	53	Reserved
24	Ground	54	Reserved
25	Not Connected	55	Reserved
26	+Terminator Power	56	Reserved
27	Ground	57	Reserved
28	Ground	58	Reserved
29	Ground	59	Reserved
30	Ground	60	Reserved

Figure 66. SCSI External Connector Pin Assignments

The following diagrams show the dimensions of the 60-pin external connector.



gure 67. 60-Pin External Connector, Front View

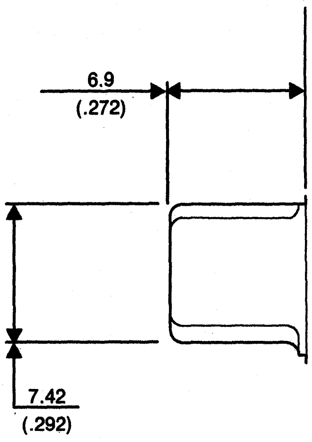


Figure 68. 60-Pin External Connector, Side View

Index