ST486DX/DX2

DATABOOK

1st EDITION

JULY 1994

USE IN LIFE SUPPORT DEVICES OR SYSTEMS MUST BE EXPRESSLY AUTHORIZED

SGS-THOMSON PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF SGS-THOMSON Microelectronics. As used herein:

1. Life support devices or systems are those which (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided with the product, can be reasonably expected to result in significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can reasonably be expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

TABLE OF CONTENTS

INTRODUCTION

♦ HIGH SPEED 3.3 VOLT VERSIONS

- Clock doubled core speeds up to 80 MHz
- Advanced 3.3 volt CMOS process technology
- I/O buffers interface to either 3.3 or 5 volt logic

♦ IMPROVED 486DX/DX2 PERFORMANCE

- Integrated FPU 10% faster than 80486DX (Power Meter Whetstone)
- 40 and 50 MHz bus speeds for fast local bus systems

♦ INDUSTRY STANDARD 486 COMPATIBILITY

- 486DX socket and instruction set compatible
- Runs DOS, Windows, OS/2, UNIX
- Standard 168-pin PGA or 208-pin QFP package

♦ ON-CHIP 8-KBYTE WRITE-BACK CACHE

- Up to 15% higher performance than write-through (PC Bench 8.0, 80 MHz)
- Industry-wide write-back chipset support
- Burst-mode write capability
- Configurable as write-back or write-through

♦ ADVANCED POWER MANAGEMENT

- Fast SMI interrupt with separate memory space
- Fully static design permits dynamic clock control
- Software or hardware initiated low-power suspend mode
- Automatic FPU power-down mode

The SGS THOMSON ST486DX/DX2™ 3 and 5 volt CPUs are advanced, 486DX/DX2 compatible processors. These CPUs incorporate an on-chip 8-KByte write-back cache and an integrated math coprocessor.

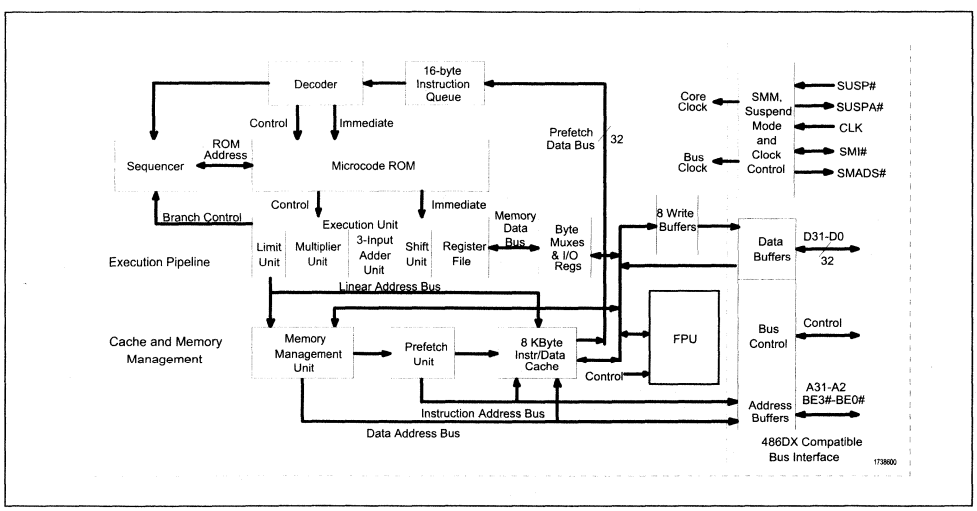
The high speed 3.3 volt ST486DX-V/DX2-V family enables clock speeds up to 80 MHz. The advanced 3.3 volt process technology power consumption is less than half the power consumption of standard 5 volt CPUs. Data, address and control pins are designed for either 3.3 or 5volt operation.

The on-chip, write-back cache allows up to 15% higher performance by eliminating unnecessary external write cycles. On traditional write-through CPUs, these external write cycles can create bus bottlenecks affecting system-wide performance.

The integrated floating point unit, based on SGS THOMSON's FasMath architecture, improves performance up to 10% over the 80486DX as measured using Power Meter Whetstone test.

These processors are designed to meet the power management requirements in the newest generation of low-power desktops and notebooks. Power is saved not only by using 3.3 volt power, but by taking advantage of advanced power management features such as static circuitry, SMM, and automatic FPU power-down. Fast entry and exit of SMM allows frequent use of the SMM feature without noticeable performance degradation.

This CPU family maintains compatibility with the installed base of x86 software and provides essential socket compatibility with the 486DX/DX2.



GENERAL INDEX

LIST OF FIGURES

Figure Name **Page Number**

LIST OF FIGURES (cont'd)

LIST OF TABLES

Figure Name **Page Number**

LIST OF TABLES (cont'd)

Figure Name **Page Number**

LIST OF TABLES (cont'd)

Figure Name **Page Number**

ST486DX/DX2 3 and 5Volt CPUs
PRODUCT OVERVIEW

1.0 PRODUCT OVERVIEW

The SGS THOMSON ST486DX™ 3.3 and 5-volt microprocessors are advanced 486DX/DX2 microprocessors. The ST486DX CPU operates at the same speed as the external bus and the ST486DX2 CPU operates at twice the external bus speed. The "ST486DX/DX2" designation refers to either the ST486DX or ST486DX2 microprocessor. The "-V" suffix indicates the CPU operates on a 3.3 volt power supply.

The CPUs in the ST486DX/DX2-V family are high speed 3.3-volt CPUs attaining clock-doubled core speeds of up to 80 MHz. Use of the 3.3v CPUs reduces power consumption by more than half compared to using the standard 5 volt CPUs. Designed into the 3.3-volt family is the ability for the data, address and control pins to interface to either 3-volt or 5-volt logic.

The ST486DX/DX2 8-KByte cache can be configured to run in traditional write-through mode or in the higher performance write-back mode. Write-back mode eliminates unnecessary external memory write cycles offering up to 15% higher overall performance (80 MHz, PC Bench 8.0) than write-through mode.

The ST486DX/DX2 supports 8, 16 and 32-bit data types and operates in real, virtual 8086 and protected modes. The CPU can access up to 4 GBytes of physical memory using a 32-bit burst mode bus. Floating point instructions are parallel processed using an on-chip math coprocessor.

The ST486DX/DX2 CPUs are ideal design solutions for low-powered "Green PC" desktops as well as portable computers. These microprocessor typically draw only 450 µA, while the input clock is stopped in suspend mode, due to their static design. System Management Mode (SMM) allows the implementation of transparent system power management or the software emulation of I/O peripheral devices.

A list of ST486DX/DX2 3-volt and 5-volt parts, including their operating frequency, and package types are listed on page A-2 in the Appendix of this manual.

1.1 Clock-Doubled CPU Core

The clock-doubled ST486DX2 CPU core operates at twice the frequency of the external clock input, while continuing to operate the bus interface at the external clock frequency. This configuration provides high frequency CPU performance without requiring a high speed interface to external memory.

The ST486DX2 provides up to 1.8 times the performance of a 486DX at the same external clock frequency. This level of performance is achieved by doubling the frequency of the input clock and using the resulting signal to drive the CPU core. To further enhance this architecture, the ST486DX2 reduces the performance penalty of slow external memory accesses through use of an on-chip write-back cache and eight write buffers.

The CPU core consists of a five-stage pipeline optimized for minimal instruction cycle times and includes all necessary hardware interlocks to permit successive instruction execution overlap. The execution stage of the pipeline executes simple but frequently used instructions in a single clock cycle and the hardware multiplier executes 16-bit integer multiplies in only three clocks.

1.2 On-Chip Write-Back Cache

The ST486DX/DX2 on-chip cache can be configured to run in traditional write-through mode or in a higher performance write-back mode. The write-back cache mode was specifically designed to optimize performance of the CPU core by eliminating bus bottlenecks caused by unnecessary external write cycles. This write-back architecture is especially effective in improving performance of the clock-doubled ST486DX2 CPU.

Traditional write-through cache architectures require that all writes to the cache also update external memory simultaneously. These unnecessary write cycles create bottlenecks which result in CPU stalls and adversely impact performance. In contrast, a write-back architecture allows data to be written to the cache without updating external memory. With a write-back cache, external write cycles are only required when a cache miss occurs, a modified line is replaced in the cache, or when an external bus master requires access to data.

The ST486DX/DX2 cache is an 8-KByte unified instruction and data cache implemented using a four-way set associative architecture and a least recently used (LRU) replacement algorithm. The cache is designed for optimum performance in write-back mode, however, the cache can be operated in write-through mode. The cache line size is 16 bytes and new lines are only allocated during memory read cycles. Valid status is maintained on a 16-byte cache line basis, but modified or "dirty" status for write-back mode is maintained on a 4-byte (double-word) basis. Therefore, only the double-words that have been modified are written back to external memory when a line is replaced in the cache. The CPU core can access the cache in a single internal clock cycle for both reads and writes.

1.3 FPU Operations

Since the FPU is resident within the CPU, the overhead associated with external math coprocessor cycles is eliminated. If the FPU is not in use, the FPU is automatically powered down. This feature reduces overall power consumption.

The integrated FPU results in the addition of two new pins FERR# (replaces ERROR#) and IGNNE#.

1.4 System Management Mode

System Management Mode (SMM) provides an additional interrupt and a separate address space that can be used for system power management or software transparent emulation of I/O peripherals. SMM is entered using the System Management Interrupt (SMI#) or SMINT instruction. While running in isolated SMM address space, the SMI interrupt routine can execute without interfering with the operating system or application programs.

After entering SMM, portions of the CPU state are automatically saved. Program execution begins at the base of SMM address space. The location and size of the SMM memory are programmable within the ST486DX/DX2. Eight SMM instructions have been added to the 486 instruction set that permit software entry into

SMM, as well as saving and restoring the total CPU state when in SMM mode.

1.5 Power Management

The ST486DX/DX2 power management features allow for a dramatic improvement in battery life over systems designed with non-static 486 processors. During suspend mode the typical current consumption is less than 1 percent of the full operation current.

Suspend mode is entered by either a hardware or a software initiated action. Using the hardware method to initiate suspend mode involves a two-pin handshake between the SUSP# and SUSPA# signals. The software can initiate suspend mode through the execution of the HALT instruction. Once in suspend mode, the ST486DX/DX2 power consumption is further reduced by stopping the external clock input. The resulting current draw is less than 500 µA. Since the ST486DX/DX2 is static, no internal data is lost when the clock is stopped.

1.6 Signal Summary

The ST486DX/DX2 signal set includes five cache interface signals, two coprocessor interface signals, two power management signals, and two system management mode signals. Refer to Chapter 3 for a detailed description of the set.

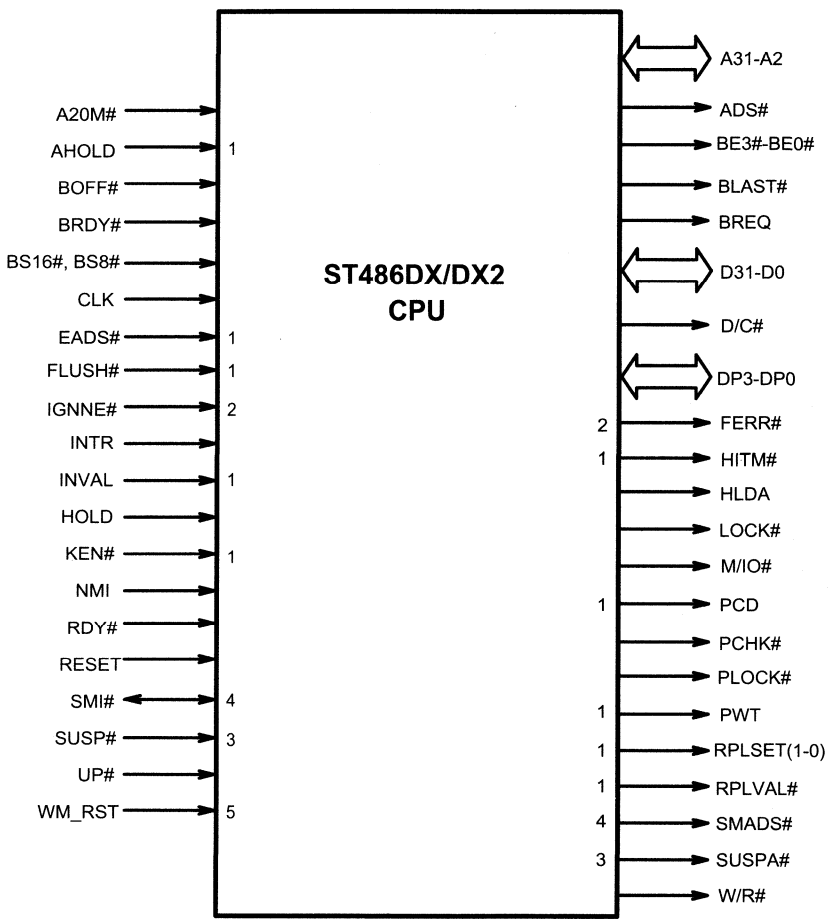


Figure 1 - 1. ST486DX/DX2 Input and Output Signals

PROGRAMMING INTERFACE

2.0 PROGRAMMING INTERFACE

In this chapter, the internal operations of the ST486DX/DX2 are described mainly from an application programmer's point of view. Included in this chapter are descriptions of processor initialization, the register set, memory addressing, various types of interrupts and the shutdown and halt process. Included is an overview of real, virtual 8086, and protected operating modes. The FPU operations are described separately at the end of this chapter.

2.1 Processor Initialization

The ST486DX/DX2 is initialized when the RESET signal is asserted. The processor is placed in real mode and the registers listed in Table 2-1 (Page 20) are set to their initialized values. RESET invalidates and disables the ST486DX/DX2 cache, and turns off paging. When RESET is asserted, the ST486DX/DX2 terminates all local bus activity and all internal execution. During the entire time that RESET is asserted, the internal pipeline is flushed and no instruction execution or bus activity occurs.

Approximately 150 to 250 external clock cycles (additional 2^{20} + 60 if self-test is requested) after RESET is negated, the processor begins executing instructions at the top of physical memory (address location FFFF FFF0h). When the first intersegment JUMP or CALL is executed, address lines A31-A20 are driven low for code segment-relative memory access cycles. While A31-A20 are low, the ST486DX/DX2 executes instructions only in the lowest 1 MByte of physical address space until system-specific initialization occurs via program execution.

Table 2 - 1. Initialized Register Controls

REGISTER	REGISTER NAME	INITIALIZED CONTENTS	COMMENTS
EAX	Accumulator	xxxx xxxxh	0000 0000h indicates self-test passed
EBX	Base	xxxx xxxxh	
ECX	Count	xxxx xxxxh	
EDX	Data	xxxx 0400 + Device ID	Device ID = 80h.
EBP	Base Pointer	xxxx xxxxh	
ESI	Source Index	xxxx xxxxh	
EDI	Destination Index	xxxx xxxxh	
ESP	Stack Pointer	xxxx xxxxh	
EFLAGS	Flag Word	0000 0002h	
EIP	Instruction Pointer	0000 FFF0h	
ES	Extra Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
CS	Code Segment	F000h	Base address set to FFFF 0000h. Limit set to FFFFh.
SS	Stack Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
DS	Data Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
FS	Extra Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
GS	Extra Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
IDTR	Interrupt Descriptor Table Register	Base = 0, Limit = 3FFh	
CR0	Machine Status Word	6000 0010h	
CCR1	Configuration Control 1	00h	
CCR2	Configuration Control 2	00h	
CCR3	Configuration Control 3	00h	
SMAR	SMM Address Region	0000h	
DIR0	Device Identification 0	ST486DX = 1Ah ST486DX2 = 1Bh	
DIR1	Device Identification 1	Step ID + Revision ID	
DR7	Debug Register 7	0000 0400h	
Note: x = Undefined value			

2.2 Instruction Set Overview

The ST486DX/DX2 instruction set can be divided into eight types of operations:

Arithmetic
Bit Manipulation
Control Transfer
Data Transfer
Floating Point
High-Level Language Support
Operating System Support
Shift/Rotate
String Manipulation.

All ST486DX/DX2 instructions operate on as few as 0 operands and as many as 3 operands. An NOP instruction (no operation) is an example of a 0 operand instruction. Two operand instructions allow the specification of an explicit source and destination pair as part of the instruction. These two operand instructions can be divided into eight groups according to operand types:

Register to Register
Register to Memory
Memory to Register
Memory to Memory
Register to I/O
I/O to Register
Immediate Data to Register
Immediate Data to Memory.

An operand can be held in the instruction itself (as in the case of an immediate operand), in a register, in an I/O port or in memory. An immediate operand is prefetched as part of the opcode for the instruction.

Operand lengths of 8, 16, or 32 bits are supported as well as 64 or 80 bit associated with floating point instructions. Operand lengths of 8 or 32 bits are generally used when executing code written for 386- or 486-class (32-bit code) processors. Operand lengths of 8 or 16 bits are generally used when executing existing 8086 or 80286 code (16-bit code). The default length of an operand can be overridden by placing one or more instruction prefixes in front of the opcode. For example, by using prefixes, a 32-bit operand can be used with 16-bit code or a 16-bit operand can be used with 32-bit code.

Chapter 6 of this manual lists each instruction in the ST486DX/DX2 instruction set along with the associated opcodes, execution clock counts and effects on the FLAGS register.

2.2.1 Lock Prefix

The LOCK prefix may be placed before certain instructions that read, modify, then write back to memory. The prefix asserts the LOCK# signal to indicate to the external hardware that the CPU is in the process of running multiple indivisible memory accesses. The LOCK prefix can be used with the following instructions:

Bit Test Instructions (BTS, BTR, BTC)
Exchange Instructions (XADD, XCHG, CMPXCHG)
One-operand Arithmetic and Logical Instructions (DEC, INC, NEG, NOT)
Two-operand Arithmetic and Logical Instructions (ADC, ADD, AND, OR, SBB, SUB, XOR).

An invalid opcode exception is generated if the LOCK prefix is used with any other instruction, or with the above instructions when no write operation to memory occurs (i.e., the destination is a register). The LOCK prefix function may be disabled by setting the NO_LOCK bit in Configuration Control Register 1 (CCR1).

2.2.3 Register Set

There are 40 accessible registers in the ST486DX/DX2 and these registers are grouped into two sets. The application register set contains the registers frequently used by application programmers, and the system register set contains the registers typically reserved for use by operating systems programmers.

The application register set is made up of eight general purpose registers, six segment registers, a flag register and a instruction pointer register.

The system register set is made up of the remaining registers which include three control registers, four system address registers, six debug registers, six configuration registers and five test registers.

Each of the registers is discussed in detail in the following sections.

2.3.1 Application Register Set

The application register set, Figure 2-1 (Page 23), consists of the registers most often used by the applications programmer. These registers are generally accessible and are not protected from read or write access.

The **General Purpose Register** contents are frequently modified by assembly language instructions and typically contain arithmetic and logical instruction operands.

The **Segment Register** in real mode contains the base address for each segment. In protected mode the segment registers contain segment selectors. The segment selectors provide indexing for tables (located in memory) that contain the base address for each segment, as well as other memory addressing information.

The **Flag Register** contains control bits used to reflect the status of previously executed instructions. This register also contains control bits that effect the operation of some instructions.

The **Instruction Pointer** register points to the next instruction that the processor will execute. This register is automatically incremented by the processor as execution progresses.

2.3.1.1 General Purpose Registers

The general purpose registers are divided into four data registers, two pointer registers, and two index registers as shown in Figure 2-2 (Page 24).

The **Data Registers** are used by the applications programmer to manipulate data structures and to hold the results of logical and arithmetic operations. Different portions of the general data registers can be addressed by using different names. An "E" prefix identifies the complete 32-bit register. An "X" suffix without the "E" prefix identifies the lower 16 bits of the register. The lower two bytes of the register can be addressed with an "H" suffix to identify the upper byte or an "L" suffix to identify the lower byte. When a destination operand size specified by an instruction is smaller than the specified destination register, the other bytes of the destination register are not affected when the operand is written to the register.

The **Pointer and Index Registers** are listed below.

SI or ESI	Source Index
DI or EDI	Destination Index
SP or ESP	Stack Pointer
BP or EBP	Base Pointer

These registers can be addressed as 16- or 32-bit registers, with the "E" prefix indicating 32 bits. The pointer and index registers can be used as general purpose registers, however, some instructions use a fixed assignment of these registers. For example, repeated string operations always use ESI as the source pointer, EDI as the destination pointer, and ECX as a counter. The instructions using fixed registers include multiply and divide, I/O access, string operations, translate, loop, variable shift and rotate, and stack operations instructions. The ST486DX/DX2 processor implements a stack using the ESP register. This stack is accessed during the PUSH and POP instructions, procedure calls, procedure returns, interrupts, exceptions, and interrupt/exception returns. The microprocessor automatically adjusts the value of the ESP during operation of these instructions.

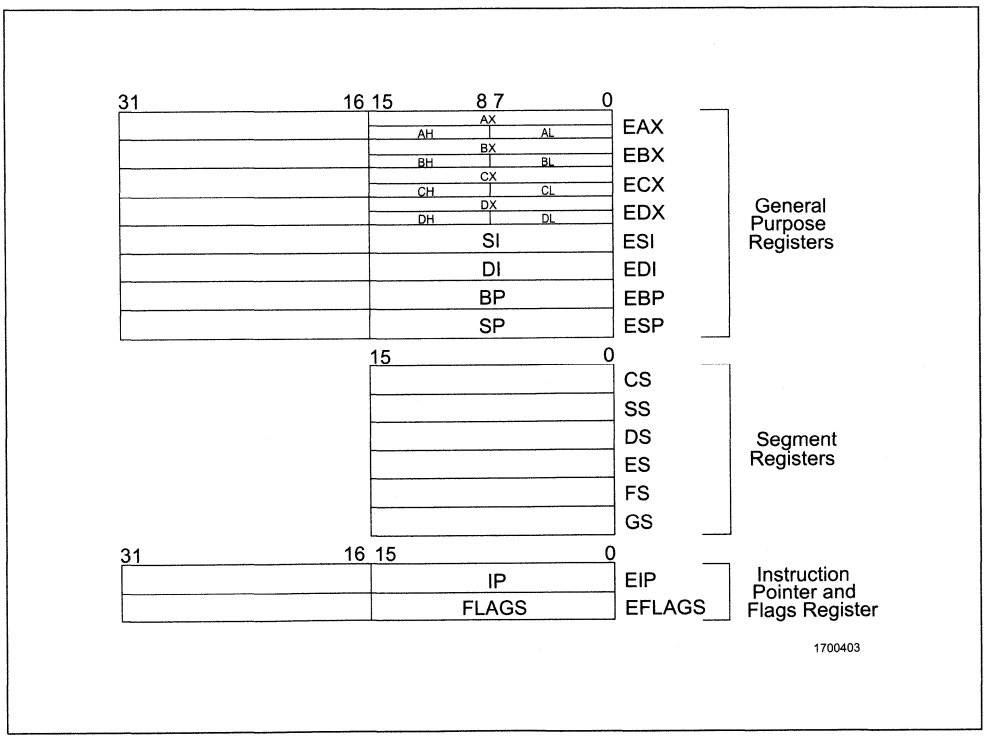


Figure 2-1. Application Register Set

The EBP register may be used to reference data passed on the stack during procedure calls. Local data may also be placed on the stack and referenced relative to BP. This register provides a mechanism to access stack data in high-level languages.

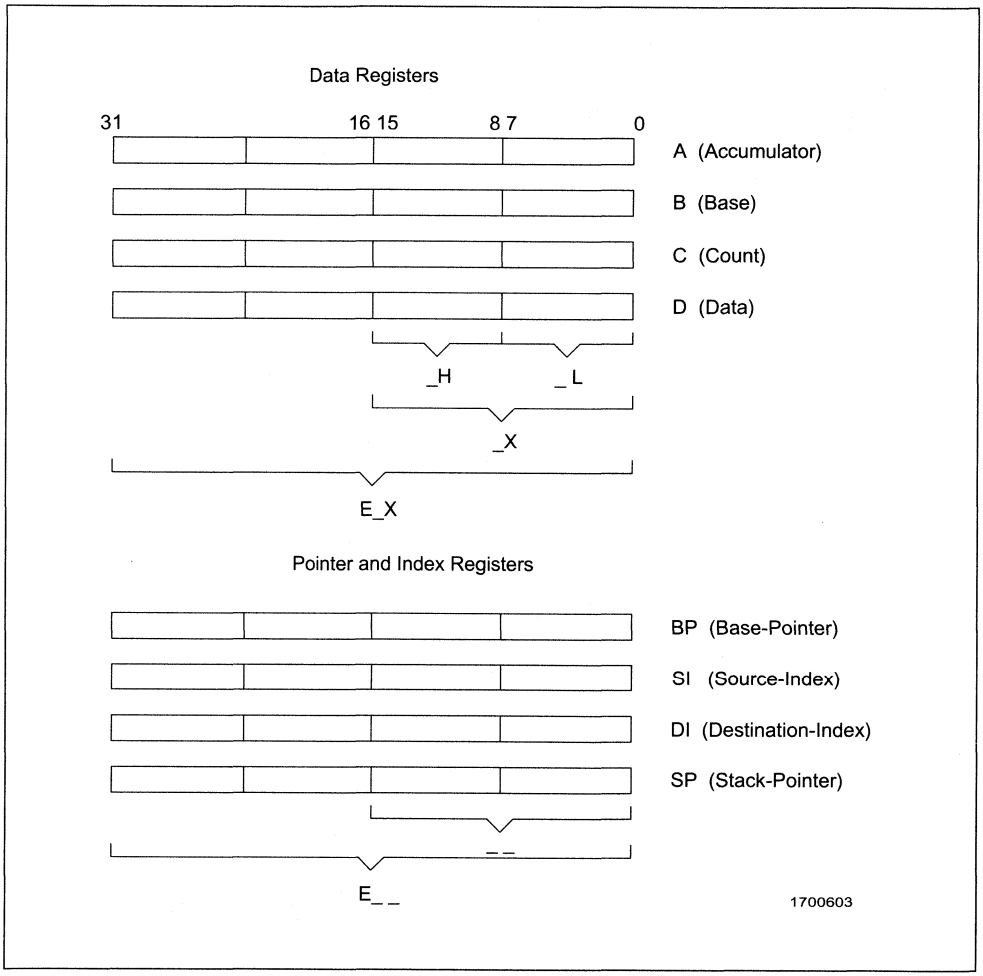


Figure 2-2. General Purpose Registers

2.3.1.2 Segment Registers and Selectors

Segmentation provides a means of defining data structures inside the memory space of the microprocessor. There are three basic types of segments: code, data, and stack. Segments are used automatically by the processor to determine the location in memory of code, data, and stack references.

There are six 16-bit segment registers:

CS	Code Segment
DS	Data Segment
ES	Extra Segment
SS	Stack Segment
FS	Additional Data Segment
GS	Additional Data Segment.

In real and virtual 8086 operating modes, a segment register holds a 16-bit segment base. The 16-bit segment base is multiplied by 16 and a 16-bit or 32-bit offset is then added to it to create a linear address. The offset size is dependent on the current address size. In real mode and in virtual 8086 mode with paging disabled, the linear address is also the physical address. In virtual 8086 mode with paging enabled, the linear address is translated to the physical address using the current page tables.

In protected mode, a segment register holds a **Segment Selector** containing a 13-bit index, a Table Indicator (TI) bit, and a two-bit Requested Privilege Level (RPL) field as shown in Figure 2-3.

The **Index** points into a descriptor table in memory and selects one of 8192 (2^{13}) segment descriptors contained in the descriptor table. A segment descriptor is an eight-byte value used to describe a memory segment by defining the segment base, the segment limit, and access control information. To address data within a segment, a 16-bit or 32-bit offset is added to the segment's base address. Once a segment selector has been loaded into a segment register, an instruction needs to specify the offset only.

The **Table Indicator** (TI) bit of the selector, defines which descriptor table the index points into. If TI=0, the index references the Global Descriptor Table (GDT). If TI=1, the index references the Local Descriptor Table (LDT). The GDT and LDT are described in more detail later in this chapter.

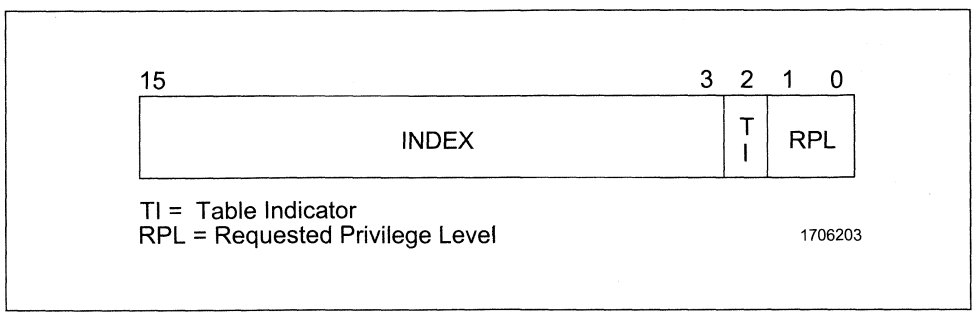


Figure 2-3. Segment Selector

The **Requested Privilege Level** (RPL) field contains a 2-bit segment privilege level (0 = most privileged, 3 = least privileged). The RPL bits are used when the segment register is loaded to determine the Effective Privilege Level (EPL). If the RPL bits indicate less privilege than the Current Program Level (CPL), the RPL overrides the CPL and the EPL is the less privileged level. If the RPL bits indicate more privilege than the program, the CPL overrides the RPL and again the EPL is the less privileged level.

When a segment register is loaded with a segment selector, the segment base, segment limit and access rights are also loaded from the descriptor table into a user-invisible or hidden portion of the segment register, i.e., cached on-chip. The CPU does not access the descriptor table again until another segment register load occurs. If the descriptor tables are modified in memory, the segment registers must be reloaded with the new selector values by the software.

The processor automatically selects a default segment register for memory references. Table 2-2 describes the selection rules. In general, data references use the selector contained in the DS register, stack references use the SS register and instruction fetches use the CS register. While some of these selections may be overridden, instruction fetches, stack operations, and the destination write of string operations cannot be overridden. Special segment override prefixes allow the use of alternate segment registers including the use of the ES, FS, and GS segment registers.

Table 2 - 2. Segment Register Selection Rules

TYPE OF MEMORY REFERENCE	IMPLIED (DEFAULT) SEGMENT	SEGMENT OVERRIDE PREFIX
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA instructions	SS	None
Source of POP, POPA, POPF, IRET, RET instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS instructions	ES	None
Other data references with effective address using base registers of:		
EAX, EBX, ECX EDX, ESI, EDI	DS	CS, ES, FS, GS, SS
EBP, ESP	SS	CS, ES, FS, GS

2.3.1.3 Instruction Pointer Register

The **Instruction Pointer** (EIP) register contains the offset into the current code segment of the next instruction to be executed. The register is normally incremented with each instruction execution unless implicitly modified through an interrupt, exception or an instruction that changes the sequential execution flow (e.g., jump, call).

2.3.1.4 Flags Register

The **Flags Register**, EFLAGS, contains status information and controls certain operations on the ST486DX/DX2 microprocessor. The lower 16 bits of this register are referred to as the FLAGS register that is used when executing 8086 or 80286 code. The flag bits are shown in Figure 2-4 and defined in Table 2-3 (Page 28).

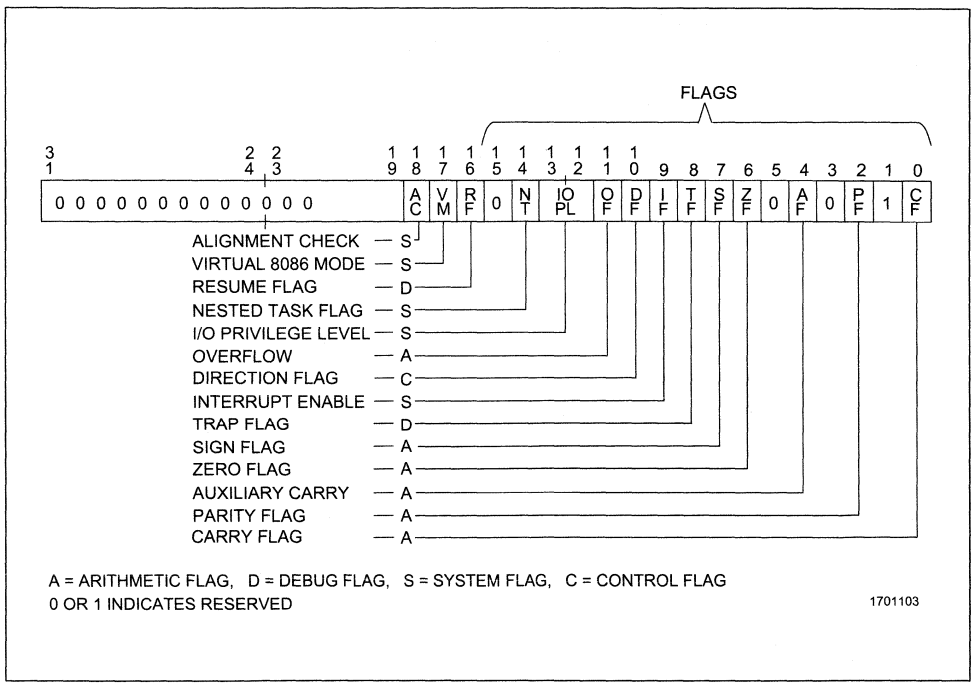


Figure 2-4. EFLAGS Register

Table 2 - 3. EFLAG Bit Definitions

BIT POSITION	NAME	FUNCTION
0	CF	Carry Flag: Set when a carry out of (addition) or borrow into (subtraction) the most significant bit of the result occurs; cleared otherwise.
2	PF	Parity Flag: Set when the low-order 8 bits of the result contain an even number of ones; cleared otherwise.
4	AF	Auxiliary Carry Flag: Set when a carry out of (addition) or borrow into (subtraction) bit position 3 of the result occurs; cleared otherwise.
6	ZF	Zero Flag: Set if result is zero; cleared otherwise.
7	SF	Sign Flag: Set equal to high-order bit of result (0 indicates positive, 1 indicates negative).
8	TF	Trap Enable Flag: Once set, a single-step interrupt occurs after the next instruction completes execution. TF is cleared by the single-step interrupt.
9	IF	Interrupt Enable Flag: When set, maskable interrupts (INTR input pin) are acknowledged and serviced by the CPU.
10	DF	Direction Flag: When cleared, DF causes string instructions to auto-increment (default) the appropriate index registers (ESI and/or EDI). Setting DF causes auto-decrement of the index registers to occur.
11	OF	Overflow Flag: Set if the operation resulted in a carry or borrow into the sign bit of the result but did not result in a carry or borrow out of the high-order bit. Also set if the operation resulted in a carry or borrow out of the high-order bit but did not result in a carry or borrow into the sign bit of the result.
12, 13	IOPL	I/O Privilege Level: While executing in protected mode, IOPL indicates the maximum current privilege level (CPL) permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O permission bit map. IOPL also indicates the maximum CPL allowing alteration of the IF bit when new values are popped into the EFLAGS register.
14	NT	Nested Task: While executing in protected mode, NT indicates that the execution of the current task is nested within another task.
16	RF	Resume Flag: Used in conjunction with debug register breakpoints. RF is checked at instruction boundaries before breakpoint exception processing. If set, any debug fault is ignored on the next instruction.
17	VM	Virtual 8086 Mode: If set while in protected mode, the microprocessor switches to virtual 8086 operation handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set by the IRET instruction (if current privilege level=0) or by task switches at any privilege level.
18	AC	Alignment Check Enable: In conjunction with the AM flag in CR0, the AC flag determines whether or not misaligned accesses to memory cause a fault. If AC is set, alignment faults are enabled.

2.3.2 System Register Set

The system register set, shown in Figure 2-5 (Page 30), consists of registers not generally used by application programmers. These registers are typically employed by system level programmers who generate operating systems and memory management programs.

The **Control Registers** control certain aspects of the ST486DX/DX2 microprocessor such as paging, coprocessor functions, and segment protection. When a paging exception occurs while paging is enabled, the control registers retain the linear address of the access that caused the exception.

The **Descriptor Table Registers** and the **Task Register** can also be referred to as system address or memory management registers. These registers consist of two 48-bit and two 16-bit registers. These registers specify the location of the data structures that control the segmentation used by the ST486DX/DX2 microprocessor. Segmentation is one available method of memory management.

The **Configuration Registers** are used to configure the ST486DX/DX2 on-chip cache operation, coprocessor interface, power management features and System Management Mode. The configuration registers also provide information on the CPU device type and revision.

The **Debug Registers** provide debugging facilities for the ST486DX/DX2 microprocessor and enable the use of data access breakpoints and code execution breakpoints.

The **Test Registers** provide a mechanism to test the contents of both the on-chip 8 KByte cache and the Translation Lookaside Buffer (TLB). The TLB is used as a cache for the tables which are used in translating linear addresses to physical addresses while paging is enabled. In the following sections, the system register set is described in greater detail.

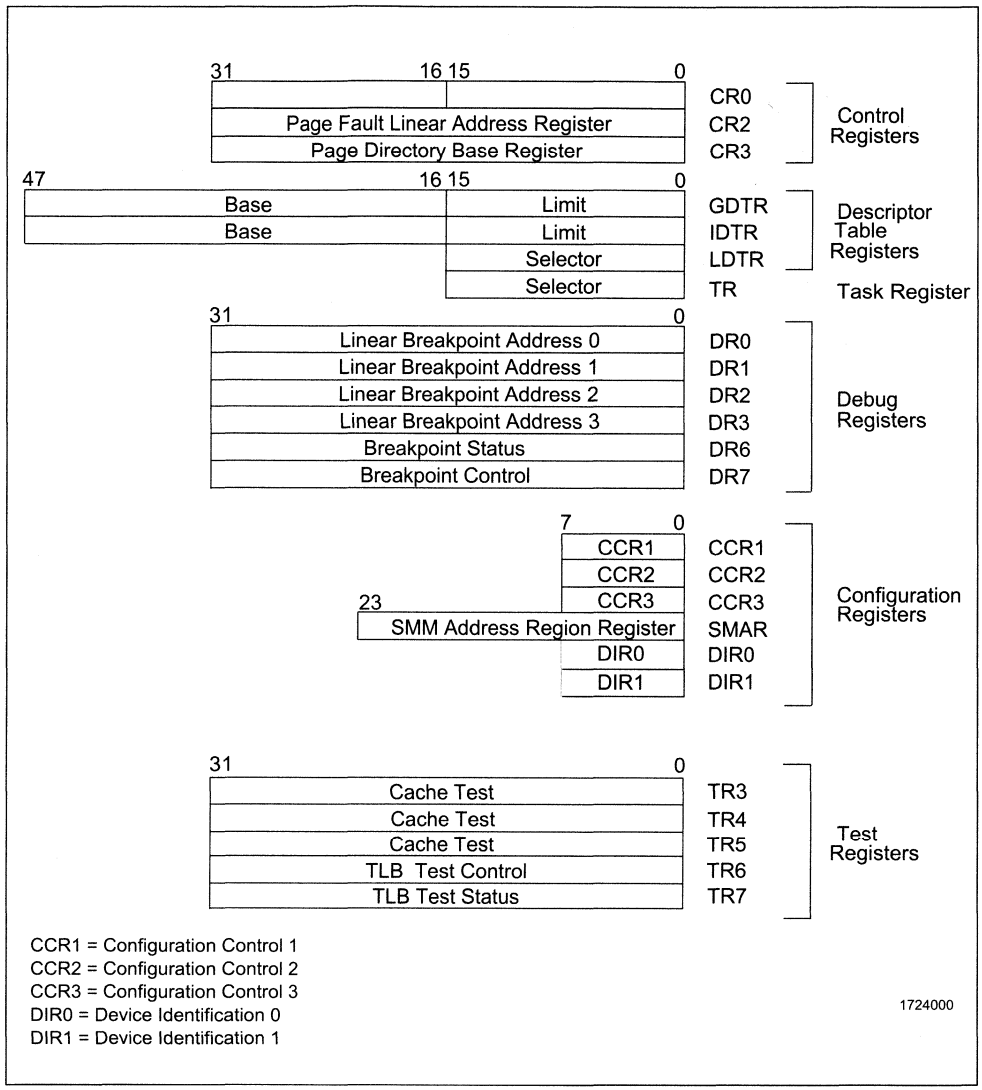


Figure 2-5. System Register Set

2.3.2.1 Control Registers

The Control Registers (CR0, CR2 and CR3), are shown in Figure 2-6. The CR0 register contains system control flags which control operating modes and indicate the general state of the CPU. The lower 16 bits of CR0 are referred to as the Machine Status Word (MSW). The CR0 bit definitions are described in Table 2-4 and Table 2-5 (Page 32). The reserved bits in CR0 should not be modified.

When paging is enabled and a page fault is generated, the CR2 register retains the 32-bit linear address of the address that caused the fault. Register CR3 contains the 20 most significant bits of the physical base address of the page directory. The page directory must always be aligned to a 4 KByte page boundary, therefore, the lower 12 bits of CR3 are not required to specify the base address.

When operating in protected mode, any program can read the control registers. However, only privilege level 0 (most privileged) programs can modify the contents of these registers.

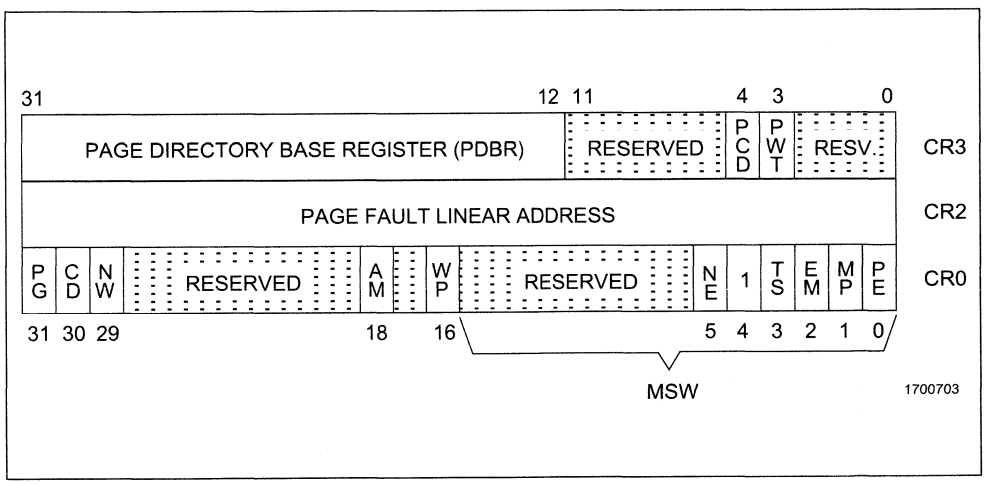


Figure 2 - 6. Control Registers

Table 2 - 4. CRO Bit Definations

BIT POSITION	NAME	FUNCTION
0	PE	Protected Mode Enable: Enables the segment based protection mechanism. If PE=1, protected mode is enabled. If PE=0, the CPU operates in real mode, with segment based protection disabled, and addresses are formed as in an 8086-class CPU.
1	MP	Monitor Processor Extension: If MP=1 and TS=1, a WAIT instruction causes Device Not Available (DNA) fault 7. The TS bit is set to 1 on task switches by the CPU. Floating point instructions are not affected by the state of the MP bit. The MP bit should be set to one during normal operations.
2	EM	Emulate Processor Extension: If EM=1, all floating point instructions cause a DNA fault 7.
3	TS	Task Switched: Set whenever a task switch operation is performed. Execution of a floating point instruction with TS=1 causes a DNA fault. If MP=1 and TS=1, a WAIT instruction also causes a DNA fault.
4	1	Reserved: Do not attempt to modify.
5	NE	Numerics Exception. NE=1 to allow FPU exceptions to be handled by interrupt 16. NE=0 if FPU exceptions are to be handled by external interrupts.
16	WP	Write Protect: Protects read-only pages from supervisor write access. WP=0 allows a read-only page to be written from privilege level 0-2. WP=1 forces a fault on a write to a read-only page from any privilege level.
18	AM	Alignment Check Mask: If AM=1, the AC bit in the EFLAGS register is unmasked and allowed to enable alignment check faults. Setting AM=0 prevents AC faults from occurring.
29	NW	Not Write-Through: If NW=1, the on-chip cache operates in write-back mode. In writeback mode, writes are issued to the external bus only for a cache miss, a line replacement of a modified line, or as the result of a cache inquiry cycle. If NW=0, the on-chip cache operates in write-through mode. In write-through mode, all writes (including cache hits) are issued to the external bus.
30	CD	Cache Disable: If CD=1, no further cache fills occur. However, data already present in the cache continues to be used if the requested address hits in the cache. The cache must also be invalidated to completely disable any cache activity.
31	PG	Paging Enable Bit: If PG=1 and protected mode is enabled (PE=1), paging is enabled.

Table 2 - 5. Effects of Various Combinations of EM, TS, and MP Bits

CR0 BIT			INSTRUCTION TYPE	
EM	TS	MP	WAIT	ESC
0	0	0	Execute	Execute
0	0	1	Execute	Execute
0	1	0	Execute	Fault 7
0	1	1	Fault 7	Fault 7
1	0	0	Execute	Fault 7
1	0	1	Execute	Fault 7
1	1	0	Execute	Fault 7

2.3.2.2 Descriptor Table Registers and Descriptors

Descriptor Table Registers

The Global, Interrupt and Local Descriptor Table Registers (GDTR, IDTR and LDTR), shown in Figure 2-7, are used to specify the location of the data structures that control segmented memory management. The GDTR, IDTR and LDTR are loaded using the LGDT, LIDT and LLDT instructions, respectively. The values of these registers are stored using the corresponding store instructions. The GDTR and IDTR load instructions are privileged instructions when operating in protected mode. The LDTR can only be accessed in protected mode.

The **Global Descriptor Table Register** (GDTR) holds a 32-bit linear base address and 16-bit limit for the Global Descriptor Table (GDT). The GDT is an array of up to 8192 8-byte descriptors. When a segment register is loaded from memory, the TI bit in the segment selector chooses either the GDT or the Local Descriptor Table (LDT) to locate a descriptor. If TI = 0, the index portion of the selector is used to locate a given descriptor within the GDT table. The contents of the GDTR are completely visible to the programmer. The first descriptor in the GDT (location 0) is not used by the CPU and is referred to as the "null descriptor". If the GDTR is loaded while operating in 16-bit operand mode, the ST486DX/DX2 accesses a 32-bit base value but the upper 8 bits are ignored resulting in a 24-bit base address.

The **Interrupt Descriptor Table Register** (IDTR) holds a 32-bit linear base address and 16-bit limit for the Interrupt Descriptor Table (IDT). The IDT is an array of 256 8-byte interrupt descriptors, each of which is used to point to an interrupt service routine. Every interrupt that may occur in the system must have an associated entry in the IDT. The contents of the IDTR are completely visible to the programmer.

The **Local Descriptor Table Register** (LDTR) holds a 16-bit selector for the Local Descriptor Table (LDT). The LDT is an array of up to 8192 8-byte descriptors. When the LDTR is loaded, the LDTR selector indexes an LDT descriptor that must reside in the Global Descriptor Table (GDT). The contents of the selected descriptor are cached on-chip in the hidden portion of the LDTR. The CPU does

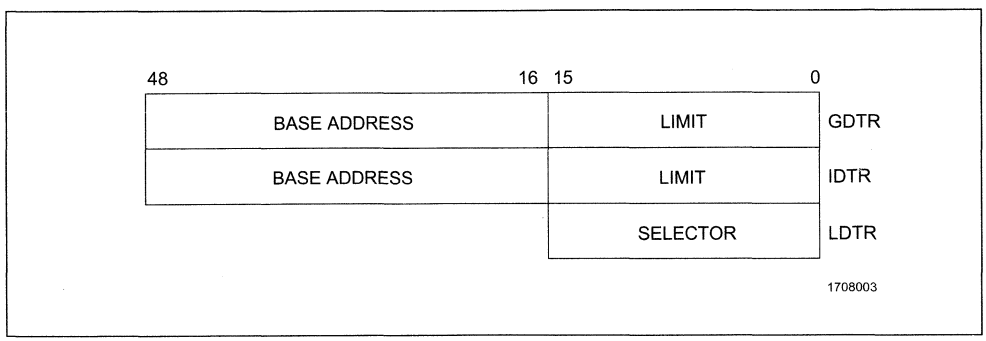


Figure 2-7. Descriptor Table Registers

not access the GDT again until the LDTR is reloaded. If the LDT descriptor is modified in memory in the GDT, the LDTR must be reloaded to update the hidden portion of the LDTR.

When a segment register is loaded from memory, the TI bit in the segment selector chooses either the GDT or the LDT to locate a segment descriptor. If TI = 1, the index portion of the selector is used to locate a given descriptor within the LDT. Each task in the system may be given its own LDT, managed by the operating system. The LDTs provide a method of isolating a given task's segments from other tasks in the system.

Descriptors

There are three types of descriptors:

- Application Segment Descriptors that define code, data and stack segments.
- System Segment Descriptors that define an LDT segment or a Task State Segment (TSS) table described later in this text.
- Gate Descriptors that define task gates, interrupt gates, trap gates and call gates.

Application Segment Descriptors can be located in either the LDT or GDT. System Segment Descriptors can only be located in the GDT. Dependent on the gate type, gate descriptors may be located in either the GDT, LDT or Interrupt Descriptor Table (IDT) described later in this text. Figure 2-8 illustrates the descriptor format for both Application Segment Descriptors and System Segment Descriptors and Table 2-6 (Page 35) lists the corresponding bit definitions.

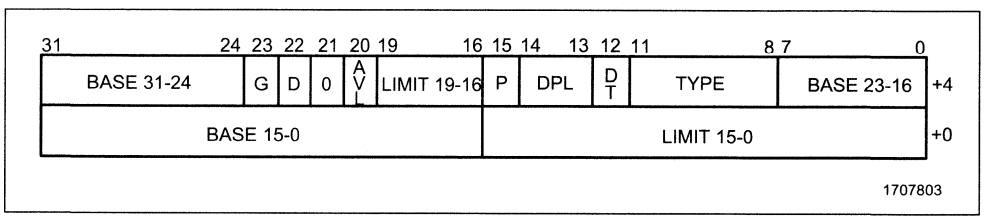


Figure 2 - 8. Application and System Segment Descriptors

Table 2 - 6. Segment Descriptor Bit Definitions

BIT POSITION	MEMORY OFFSET	NAME	DESCRIPTION
31-24 7-0 31-16	+4 +4 +0	BASE	Segment base address. 32-bit linear address that points to the beginning of the segment.
19-16 15-0	+4 +0	LIMIT	Segment limit. In real mode, the default segment limit is always 64 KBytes (0FFFFh).
23	+4	G	Limit granularity bit: 0 = byte granularity, 1 = 4 KBytes (page) granularity.
22	+4	D	Default length for operands and effective addresses. Valid for code and stack segments only: 0 = 16 bit, 1 = 32-bit.
20	+4	AVL	Segment available.
15	+4	P	Segment present.
14-13	+4	DPL	Descriptor privilege level.
12	+4	DT	Descriptor type: 0 = system, 1 = application.
11-8	+4	TYPE	Segment type. System descriptor (DT = 0): 0010 = LDT descriptor. 1001 = TSS descriptor, task not busy. 1011 = TSS descriptor, task busy.
11		E	Application descriptor (DT = 1): 0 = data, 1 = executable.
10		C/D	If E = 0: 0 = expand up, limit is upper bound of segment. 1 = expand down, limit is lower bound of segment. If E = 1: 0 = non-conforming. 1 = conforming (runs at privilege level of calling procedure).
9		R/W	If E = 0: 0 = non-writable. 1 = writable. If E = 1: 0 = non-readable 1 = readable
8		A	0 = not accessed, 1 = accessed.

Gate Descriptors provide protection for executable segments operating at different privilege levels. Figure 2-9 illustrates the format for Gate Descriptors and Table 2-7 lists the corresponding bit definitions.

Task Gate Descriptors (TGD) are used to switch the CPU's context during a task switch. The selector portion of the TGD locates a Task State Segment. TGDs can be located in the GDT, LDT or IDT tables.

Interrupt Gate Descriptors are used to enter a hardware interrupt service routine. Trap Gate Descriptors are used to enter exceptions or software interrupt service routines. Trap Gate and Interrupt Gate Descriptors can only be located in the IDT.

Call Gate Descriptors are used to enter a procedure (subroutine) that executes at the same or a more privileged level. A Call Gate Descriptor primarily defines the procedure entry point and the procedure's privilege level.

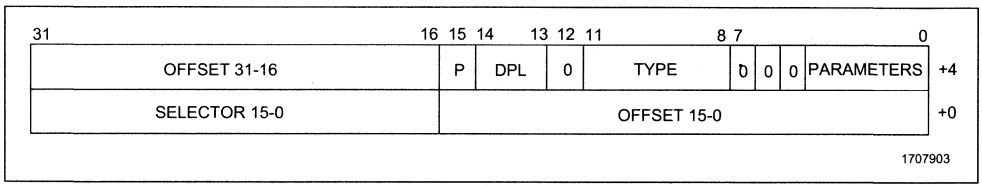


Figure 2 - 9. Gate Descriptor

Table 2 - 7. Gate Descriptor Bit Definitions

BIT POSITION	MEMORY OFFSET	NAME	DESCRIPTION
31-16 15-0	+4 +0	OFFSET	Offset used during a call gate to calculate the branch target.
31-16	+0	SELECTOR	Segment selector used during a call gate to calculate the branch target.
15	+4	P	Segment present.
14-13	+4	DPL	Descriptor privilege level.
11-8	+4	TYPE	Segment type: 0100 = 16-bit call gate0101 = task gate 0110 = 16-bit interrupt gate 0111 = 16-bit trap gate1100 = 32-bit call gate 1110 = 32-bit interrupt gate 1111 = 32-bit trap gate.
4-0	+4	PARAMETERS	Number of 32-bit parameters to copy from the caller's stack to the called procedure's stack.

2.3.2.3 Task Register

The **Task Register** (TR) holds a 16-bit selector for the current Task State Segment (TSS) table as shown in Figure 2-10. The TR is loaded and stored via the LTR and STR instructions, respectively. The TR can only be accessed during protected mode and can only be loaded when the privilege level is 0 (most privileged). When the TR is loaded, the TR selector field indexes a TSS descriptor that must reside in the Global Descriptor Table (GDT).

The contents of the selected descriptor are cached on-chip in the hidden portion of the TR.

During task switching, the processor saves the current CPU state in the TSS before starting a new task. The TR points to the current TSS. The TSS can be either a 386/486-type 32-bit TSS (Figure 2-11, Page 38) or a 286-type 16-bit TSS type (Figure 2-12, Page 39). An I/O permission bit map is referenced in the 32-bit TSS by the I/O Map Base Address.

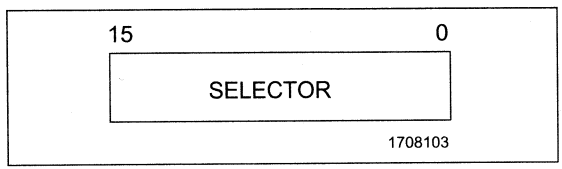


Figure 2 - 10. Task Register

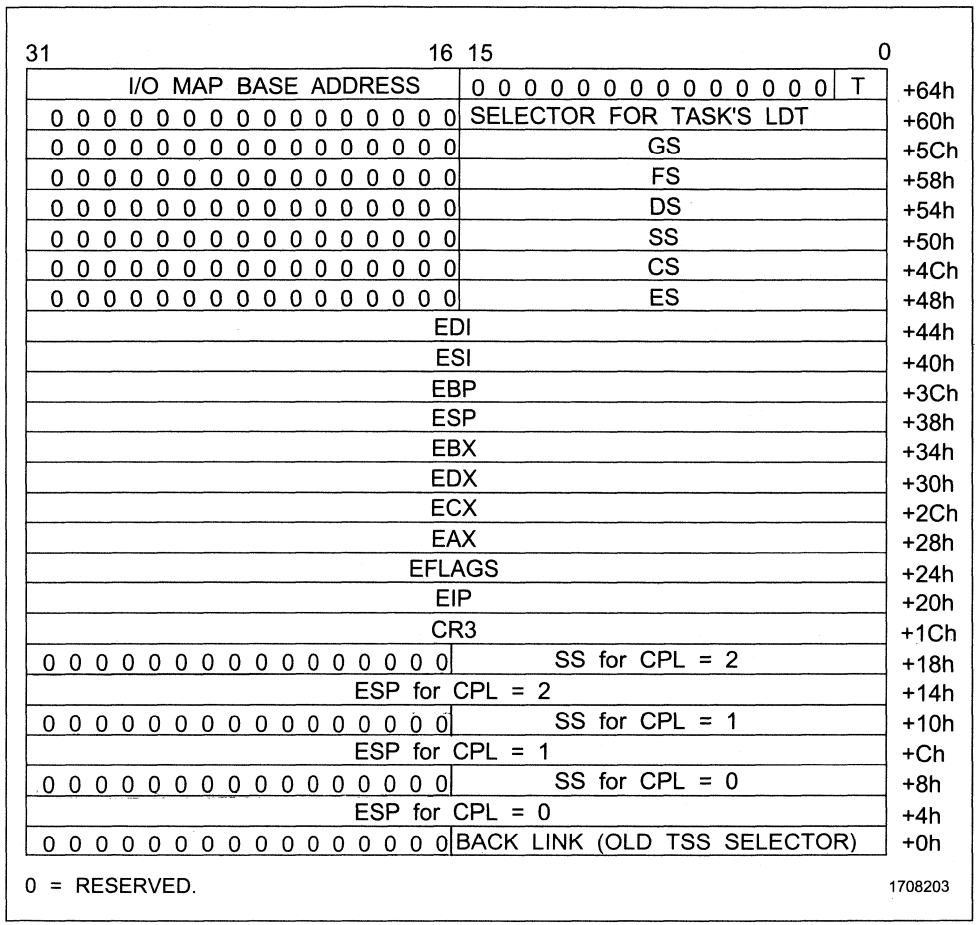


Figure 2 - 11. 32-Bit Task State Segment (TSS) Table

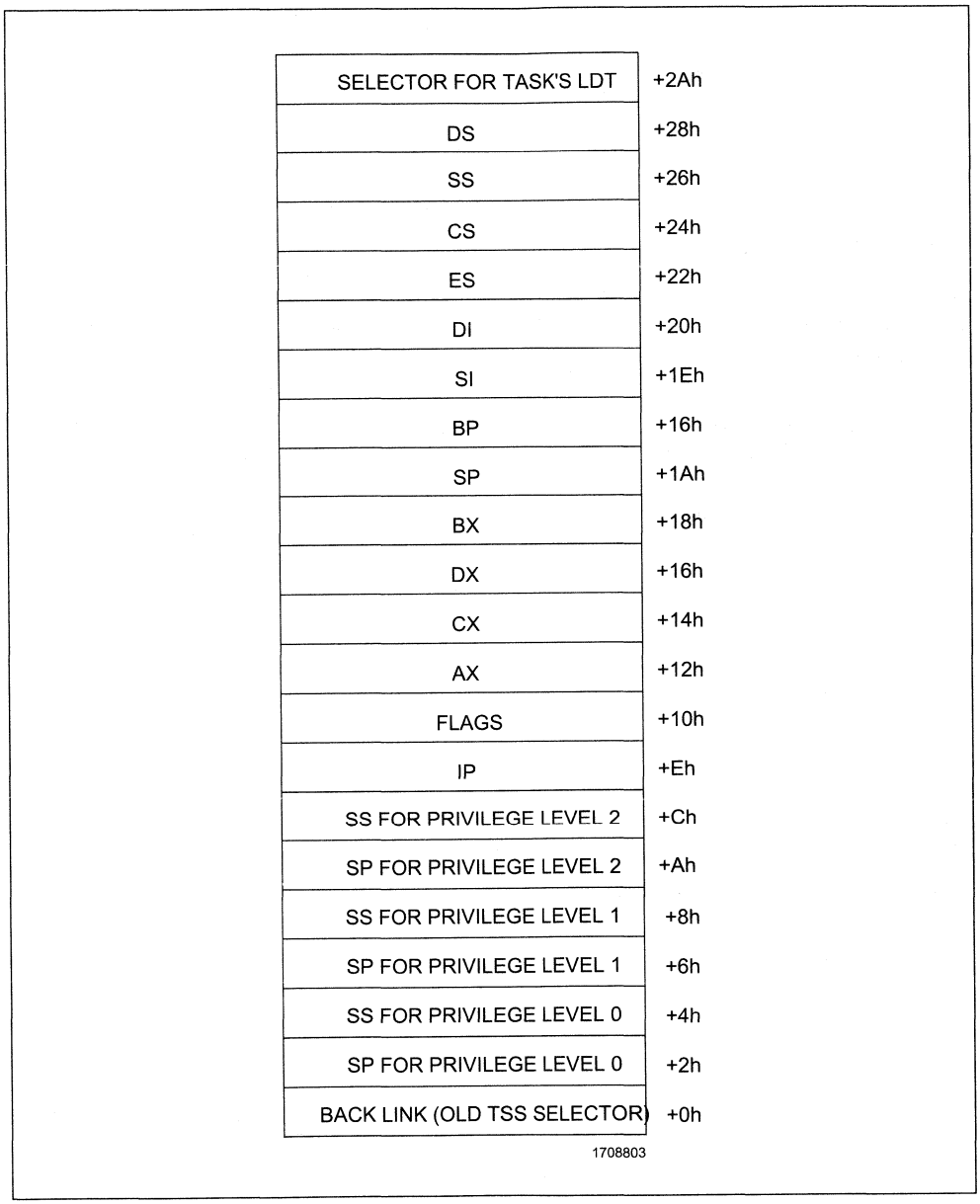


Figure 2-12. 16-Bit Task State Segment (TSS) Table

2.3.2.4 Configuration Registers

The ST486DX/DX2 provides three 8-bit Configuration Control Registers (CCR1, CCR2 and CCR3) used to control the on-chip write-back cache, the coprocessor interface pins and SMM features. The ST486DX/DX2 also provides two 8-bit internal read-only device identification registers (DIR0 and DIR1) and one 24-bit SMM Address Region Register (SMAR). The CCR, DIR, and SMAR registers exist in I/O memory space and are selected by a "register index" number as listed in (Table 2-8, Page 41).

Access to these registers is achieved by writing the index of the register to I/O port 22h. I/O port 23h is then used for data transfer. Each I/O port 23h data transfer must be preceded by an I/O port 22h register index selection, otherwise the second and later I/O port 23h operations are directed off-chip and produce external I/O cycles. If the register index number is outside the C0h-CFh, FEh-FFh range, external I/O cycles will also occur.

The CCR1 register (Figure 2-13, Page 41), (Table 2-9, Page 41) controls SMM features and enables SMM and cache interface pins.

The CCR2 register (Figure 2-14, Page 42), (Table 2-10, Page 42) is used to setup internal cache operation and enable suspend control pins.

The CCR3 register (Figure 2-17, Page 45), (Table 2-11, Page 43) controls additional SMM features.

The SMAR register (Figure 2-16, Page 44), (Table 2-12, Page 44), is used to define the location and size of the memory region associated with SMM memory space. The starting address of the SMM address region must be on a block size boundary. For example, a 128 KByte block is allowed to have a starting address of 0 KBytes, 128 KBytes, 256 KBytes, etc. The SMM block size must be defined for SMI# to be recognized.

DIR0 (Figure 2-17, Page 45), (Table 2-13, Page 45) contains an 8-bit value that defines the device type.

DIR1 (Figure 2-18, Page 46), (Table 2-14, Page 46) contains additional device type information. The upper 4 bits of DIR1 represent the stepping number of the device and the lower 4 bits of DIR1 represent the particular revision number of the stepping. Actual values for DIR0 and DIR1 are shown in Initialized Register Controls, Table 2-1 (Page 20) earlier in this chapter.

Table 2-8. Configuration Registers

REGISTER NAME	REGISTER INDEX	WIDTH
Configuration Control 1 CCR1	C1h	8 bits
Configuration Control 2 CCR2	C2h	8 bits
Configuration Control 3 CCR3	C3h	8 bits
SMM Address Region SMAR	CDh, CEh, CFh	24 bits
Device Identification 0 DIR0	FEh	8 bits
Device Identification 1 DIR1	FFh	8 bits
Note: The following register index numbers are reserved for future use: C0h through CFh and FEh, FFh.		

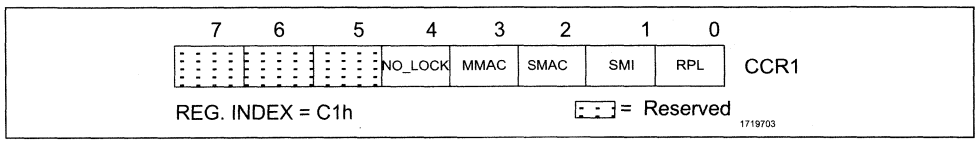


Figure 2 - 13. Configuration Control Register 1 (CCR 1)

Table 2 - 9. CCR1 Bit Defination

BIT POSITION	NAME	DESCRIPTION
0	RPL	Enable RPL Pins If = 1: Enable output pins RPLSET(1-0) and RPLVAL#. If = 0: Output pins RPLSET(1-0) and RPLVAL# float.
1	SMI	Enable SMM Pins If = 1: SMI# input/output pin and SMADS# output pin are enabled. If = 0: SMI# input pin ignored and SMADS# output pin floats.
2	SMAC	System Management Memory Access If = 1: Any access to addresses within the SMM memory space cause external bus cycles to be issued with SMADS# output active. SMI# input is ignored. If = 0: No effect on access.
3	MMAC	Main Memory Access If = 1: All data accesses which occur within an SMI service routine (or when SMAC = 1) access main memory instead of SMM memory space. If = 0: No effect on access.
4	NO_LOCK	Negate LOCK# If = 1: All bus cycles are issued with LOCK# pin negated except page table accesses. Interrupt acknowledge cycles are executed as locked cycles even though LOCK# is negated. With NO_LOCK set, previously noncacheable locked cycles are executed as unlocked cycles and therefore, may be cached. This results in higher CPU performance.
7-5	*Reserved*	
Note: Bits 4-0 are cleared to 0 at reset.		

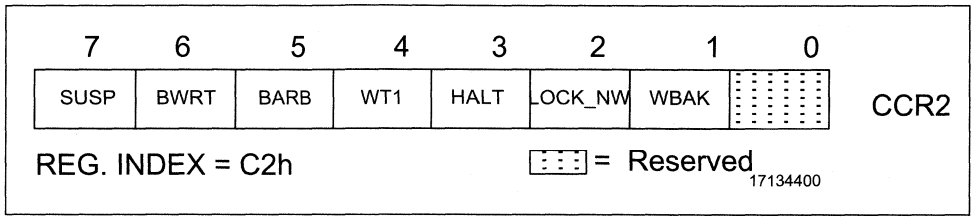


Figure 2 - 14. Configuration Control Register 2 (CCR 2)

Table 2 - 10. CCR2 Bit Definitions

BIT POSITION	NAME	DESCRIPTION
1	WBAK	EnableWrite-Back Cache Interface Pins If = 1: Enable INVAL and WM_RST input pins, and HITM# output pin. If = 0: INVAL and WM_RST input pins are ignored, and HITM# output pin floats.
2	LOCK_NW	LOCK NW Bit If = 1: Prohibits changing the state of the NW bit in CRO.
3	HALT	Suspend on HALT If = 1: CPU enters suspend mode following execution of a HALT instruction.
4	WT1	Write-Through Region 1 If = 1: Forces all writes to the 640 KBytes to 1 MByte address region that hit in the on-chip cache to be issued on the external bus.
5	BARB	Enable Cache Coherency on Bus Arbitration If = 1: Enable write-back of all dirty cache data when HOLD is requested and prior to asserting HLDA.
6	BWRT	Enable Burst Write Cycles If = 1: Enables use of 16-byte burst write-back cycles.
7	SUSP	Enable Suspend Pins If = 1: SUSP# input and SUSPA# output are enabled. If = 0: SUSP# input is ignored and SUSPA# output floats.
Note: All bits are cleared to zero at reset		

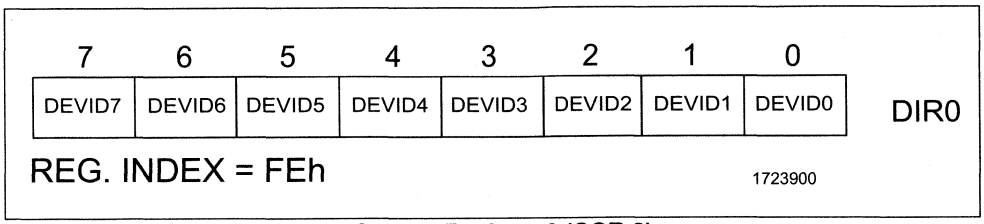


Figure 2 - 15. Configuration Control Register 3 (CCR 3)

Table 2 - 11. CCR3 Bit Definations

BIT POSITION	NAME	DESCRIPTION
0	SMI_LOCK	SMM Register Lock If = 1: the following SMM control bits cannot be modified: CCR1 bits: 1, 2, and 3 CCR3: bit 1 all SMAR bits. However, while operating within a SMI handler these SMM control bits can be modified. Once set, the SMI_LOCK bit can only br cleared by asserting the RESET pin.
1	NMIEN	NMI Enable If = 1: NMI is enabled during SMM. If = 0: NMI is not recognized during SMM.
7-2	*Reserved*	
Note: Bits 1-0 are cleared to zero at reset.		

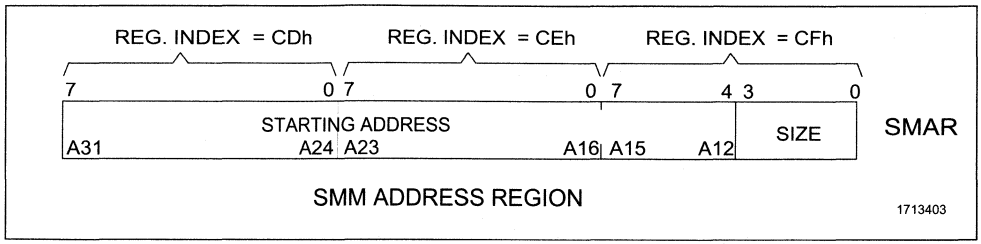


Figure 2 - 16. SMM Address Region Registers (SMAR)

Table 2 - 12. SMAR Size Field

BITS 3-0	BLOCK SIZE		BITS 3-0	BLOCK SIZE
0h	Disabled		8h	512 KBytes
1h	4 KBytes		9h	1 MBytes
2h	8 KBytes		Ah	2 MBytes
3h	16 KBytes		Bh	4 MBytes
4h	32 KBytes		Ch	8 MBytes
5h	64 KBytes		Dh	16 MBytes
6h	128 KBytes		Eh	32 MBytes
7h	256 KBytes		Fh	4 KBytes (same as 1h)

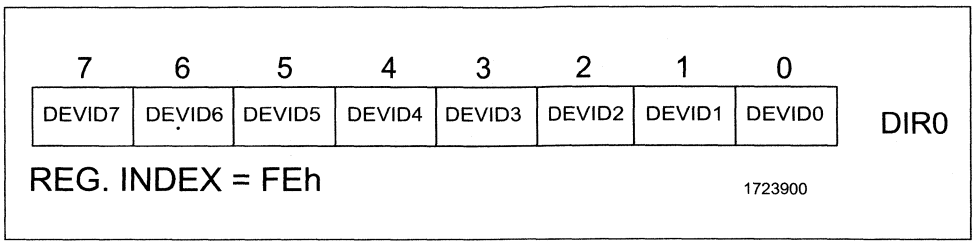


Figure 2 - 17. Device Identification Register 0 (DIRO)

Table 2 - 13. DIRO Bit Definations

BIT POSITION	NAME	DESCRIPTION
7-0	DEVID(7-0)	Device Identification: DEVID(7-0) bits define the CPU type. These bits are read only. ST486DX = 1Ah, ST486DX 2 = 1Bh.

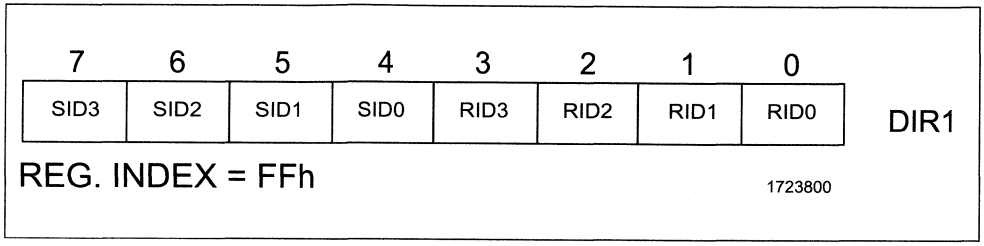


Figure 2 - 18. Device Identification Register 1 (DIR1)

Table 2 - 14. DIR1 Bit Definations

BIT POSITION	NAME	DESCRIPTION
3-0	RID(3-0)	Revision Identification: RID(3-0) bits are read only. and indicate device revision number.
7-4	SID(3-0)	Stepping Identification: RID(3-0) bits are read only. and indicate device stepping number.

2.3.2.5 Debug Registers

Six debug registers (DR0-DR3, DR6 and DR7), shown in Figure 2-19, support debugging on the ST486DX/DX2. Memory addresses loaded in the debug registers, referred to as "breakpoints", generate a debug exception when a memory access of the specified type occurs to the specified address. A breakpoint can be specified for a particular kind of memory access such as a read or a write. Code and data breakpoints can also be set allowing debug exceptions to occur whenever a given data access (read or write) or code access (execute) occurs. The size of the debug target can be set to 1, 2, or 4 bytes. The debug registers are accessed via MOV instructions which can be executed only at privilege level 0.

The Debug Address Registers (DR0-DR3) each contain the linear address for one of four possible breakpoints. Each breakpoint is further specified by bits in the Debug Control Register (DR7). For each breakpoint address in DR0-DR3, there are corresponding fields L, R/W, and LEN in DR7 that specify the type of memory access associated with the breakpoint.

The R/W field can be used to specify instruction execution as well as data access breakpoints. Instruction execution breakpoints are always taken before execution of the instruction that matches the breakpoint.

The Debug Status Register (DR6) reflects conditions that were in effect at the time the debug exception occurred. The contents of the DR6 register are not automatically cleared by the processor after a debug exception occurs and, therefore, should be cleared by software at the appropriate time. Table 2-15 (Page 48) lists the field definitions for the DR6 and DR7 registers.

Code execution breakpoints may also be generated by placing the breakpoint instruction (INT 3) at the location where control is to be regained. The single-step feature may be enabled by setting the TF flag in the EFLAGS register. This causes the processor to perform a debug exception after the execution of every instruction.

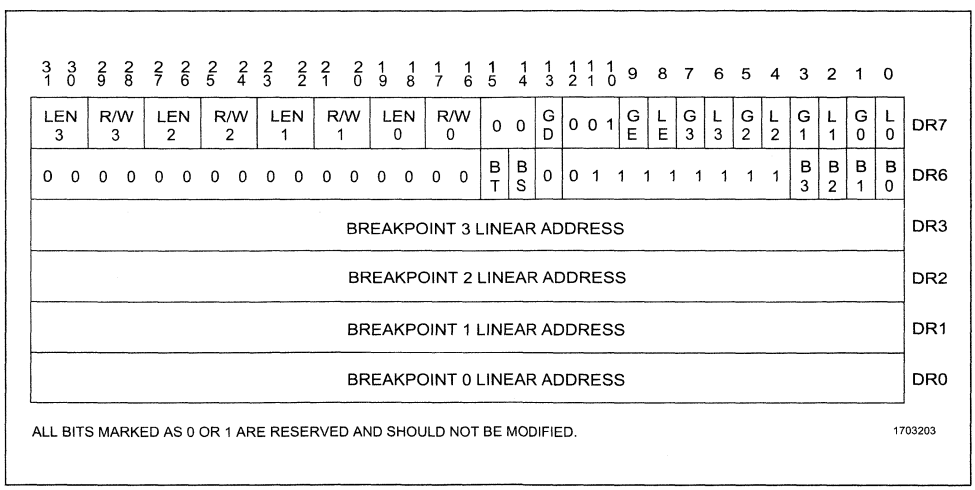


Figure 2 - 19. Debug Registers

Table 2 - 15. DR6 and DR7 Field Definitions

REGISTER	FIELD	NUMBER OF BITS	DESCRIPTION
DR6	Bi	1	Bi is set by the processor if the conditions described by DRi, R/Wi, and LENi occurred when the debug exception occurred, even if the breakpoint is not enabled via the Gi or Li bits.
	BT	1	BT is set by the processor before entering the debug handler if a task switch has occurred to a task with the T bit in the TSS set.
	BS	1	BS is set by the processor if the debug exception was triggered by the single-step execution mode (TF flag in EFLAGS set).
DR7	R/Wi	2	Applies to the DRi breakpoint address register: 00 - Break on instruction execution only 01 - Break on data writes only 10 - Not used 11 - Break on data reads or writes.
	LENi	2	Applies to the DRi breakpoint address register: 00 - One byte length 01 - Two byte length 10 - Not used 11 - Four byte length.
	Gi	I	If set to a 1, breakpoint in DRi is globally enabled for all tasks and is not cleared by the processor as the result of a task switch.
	Li	1	If set to a 1, breakpoint in DRi is locally enabled for the current task and is cleared by the processor as the result of a task switch.
	GD	1	Global disable of debug register access. GD bit is cleared whenever a debug exception occurs.

2.3.2.6 Test Registers

The five test registers, shown in Figure 2-20, are used in testing the CPU's Translation Look-aside Buffer (TLB) and on-chip cache. TR6 and TR7 are used for TLB testing, and TR3-TR5 are used for cache testing. Table 2-16 (Page 50) and Table 2-17 (Page 51) list the bit definitions for the TR6 and TR7 registers.

TLB Test Registers

The ST486DX/DX2 TLB is a four-way set associative memory with eight entries per set. Each TLB entry consists of a 24-bit tag and 20-bit data. The 24-bit tag represents the high-order 20 bits of the linear address, a valid bit, and three attribute bits. The 20-bit data portion represents the upper 20 bits of the physical address that corresponds to the linear address.

The TLB Test Control Register (TR6) contains a command bit, the upper 20 bits of a linear address, a valid bit and the attribute bits used in the test operation. The contents of TR6 is used to create the 24-bit TLB tag during both write and read (TLB lookup) test operations. The command bit defines whether the test operation is a read or a write.

The TLB Test Data Register (TR7) contains the upper 20 bits of the physical address (TLB data field), three LRU bits and a control bit. During TLB write operations, the physical address in TR7 is written into the TLB entry selected by the contents of TR6. During TLB

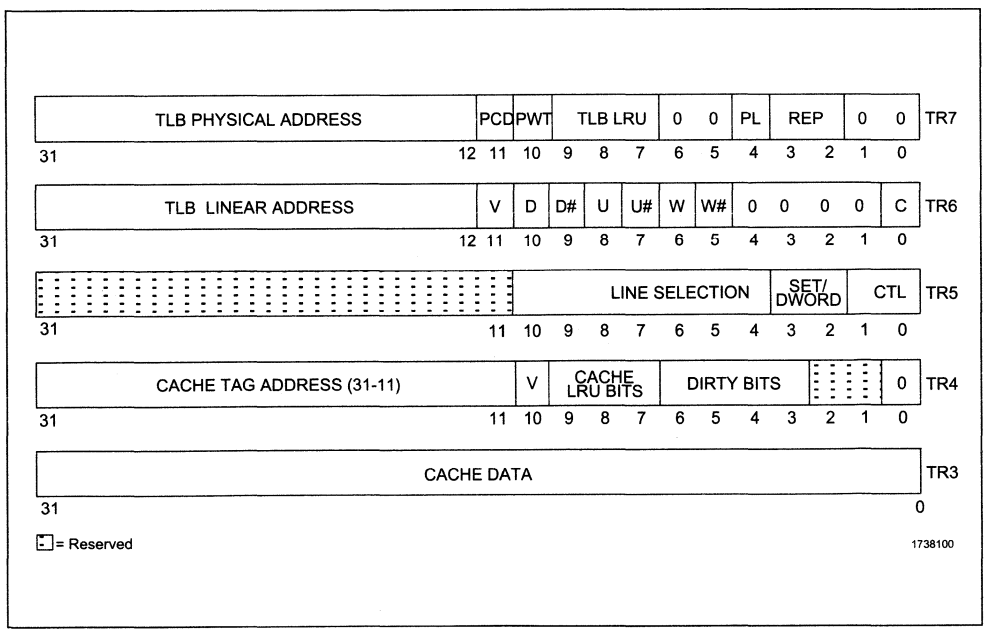


Figure 2 - 20. Test Registers

Table 2 - 16. TR6 and TR7 Bit Definitions

REGISTER NAME	BIT POSITION	DESCRIPTION
TR6	31-12	Linear address. TLB lookup: The TLB is interrogated per this address. If one and only one match occurs in the TLB, the rest of the fields in TR6 and TR7 are updated per the matching TLB entry. TLB write: A TLB entry is allocated to this linear address.
	11	Valid bit (V). TLB write: If set, indicates that the TLB entry contains valid data. If clear, target entry is invalidated.
	10-9	Dirty attribute bit and its complement (D, D#). Refer to Table 2-17 (Page 2-33).
	8-7	User/supervisor attribute bit and its complement (U, U#). Refer to Table 2-17 (Page 2-33).
	6-5	Read/write attribute bit and its complement (R, R#). Refer to Table 2-17 (Page 2-33).
	0	Command bit (C). If = 0: TLB write. If = 1: TLB lookup.
TR7	31-12	Physical address. TLB lookup: data field from the TLB. TLB write: data field written into the TLB.
	11	Page-level cache disable bit (PCD). Corresponds to the PCD bit of a page table entry.
	10	Page-level cache write-through bit (PWT). Corresponds to the PWT bit of a page table entry.
	9-7	LRU bits. TLB lookup: LRU bits associated with the TLB entry prior to the TLB lookup. TLB write: ignored.
	4	PL bit. TLB lookup: If = 1, read hit occurred. If = 0, read miss occurred. TLB write: If = 1, REP field is used to select the set. If = 0, the pseudo-LRU replacement algorithm is used to select the set.
	3-2	Set selection (REP). TLB lookup: If PL = 1, set in which the tag was found. If PL = 0, undefined data. TLB write: If PL = 1, selects one of the four sets for replacement. If PL = 0, ignored.

Table 2 - 17. TR6 Attribute Bit Pairs

BIT (D, U or R)	BIT COMPLEMENT (D#, U#, or R#)	EFFECT ON TLB LOOKUP	EFFECT ON TLB WRITE
0	0	Do not match.	Undefined.
0	1	Match if D, U or R bit is a 0.	Clear the bit.
1	0	Match if D, U or R bit is a 1.	Set the bit.
1	1	Match if D, U or R bit is either a 1 or 0.	Undefined.

lookup operations, the TLB data selected by the contents of TR6 is loaded into TR7.

Cache Test Registers

The ST486DX/DX2 8-KByte on-chip cache is a four-way set associative memory that can be configured as either write-back or write-through. Each cache set contains 128 entries. Each entry consists of a 21-bit tag address, a 16-byte data field, a valid bit, and four dirty bits.

The 21-bit tag represents the high-order 21 bits of the physical address. The 16-byte data represents the 16 bytes of data currently in memory at the physical address represented by the tag. The valid bit indicates whether the data bytes in the cache actually contain valid data. The four dirty bits indicate if the data bytes in the cache have been modified internally without updating external memory (write-back configuration). Each dirty bit indicates the status for one double-word (4 bytes) within the 16-byte data field.

For each line in the cache, there are three LRU bits that indicate which of the four sets was most recently accessed. A line is selected using bits 0 - 4 of the physical address. Figure 2-21 illustrates the ST486DX/DX2 cache architecture.

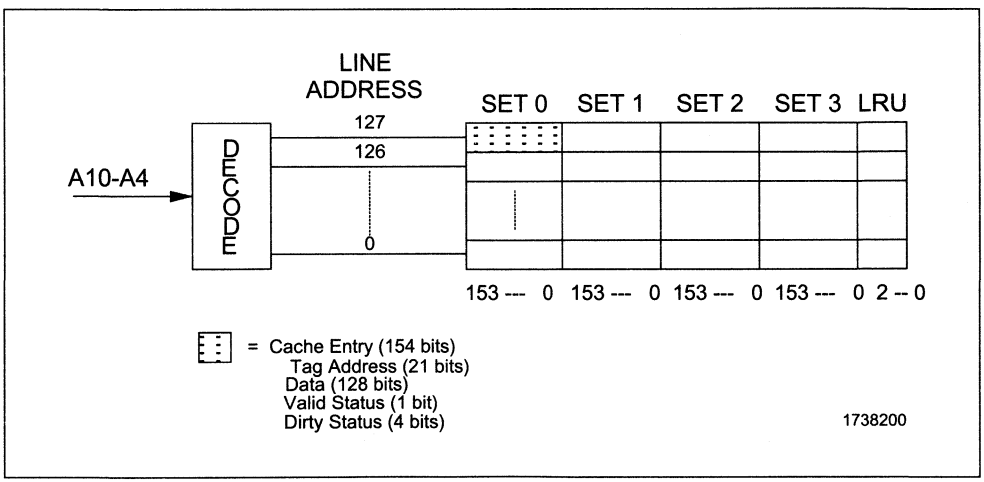


Figure 2-21. ST486DX/DX2 Cache Architecture

The ST486DX/DX2 contains three test registers that allow testing of its internal cache. Bit definitions for the cache test registers are shown in Table 2-18 (Page 53). Using these registers, cache writes and reads may be performed.

Cache test writes cause the data in the cache fill buffer to be written to the selected set and entry in the cache. Data must be written to TR3 (32--bit register) four times in order to fill the cache fill buffer. Once the cache fill buffer has been loaded, a cache test write can be performed. For data to be written to the allocated entry, the valid bit for the entry must be set prior to the write of the data.

Cache test reads cause the data in the selected set and entry to be loaded into the cache flush buffer. Once the buffer has been loaded, data must be read from TR3 four times in order to empty the cache flush buffer. For proper operation, cache tests should be performed only when the cache is disabled (CD bit in CR0 = 1).

Table 2 - 18. TR3-TR5 Bit Definitions

REGISTER NAME	BIT POSITION	DESCRIPTION
TR3	31-0	Cache data. Flush buffer read: data accessed from the cache flush buffer. Fill buffer write: data to be written into the cache fill buffer.
TR4	31-11	Upper Tag Address. Cache read: upper 21 bits of tag address of the selected entry. Cache write: data written into the upper 21 bits of the tag address of the selected entry.
	10	Valid Bit. Cache read: valid bit for the selected entry. Cache write: data written into the valid bit for the selected entry.
	9-7	LRU Bits. Cache read: the LRU bits for the selected line. xx1 = Set 0 or Set 1 most recently accessed. xx0 = Set 2 or Set 3 most recently accessed. x1x = Most recent access to Set 0 or Set 1 was to Set 0. x0x = Most recent access to Set 0 or Set 1 was to Set 1. 1xx = Most recent access to Set 2 or Set 3 was to Set 2. 0xx = Most recent access to Set 2 or Set 3 was to Set 3. Cache write: ignored.
	6-3	Dirty Bits. Cache read: the dirty bits for the selected entry (one bit per dword). Cache write: data written into the dirty bits for the selected entry.
TR5	10-4	Line Selection. Physical address bits 10-4 used to select one of 128 lines.
	3-2	Set/DWord Selection. Cache read: selects which of the four sets is used as the source for data transferred to the cache flush buffer. Cache write: selects which of the four sets is used as the destination for data transferred from the cache fill buffer. Flush buffer read: selects which of the four dwords in the flush buffer is loaded into TR3. Fill buffer write: selects which of the four dwords in TR3 is written to the fill buffer.
	1-0	Control Bits. If = 00: flush read or fill buffer write. Writing to TR3 fill buffer write. Reading TR3 initiates flush buffer read. If = 01: cache write. If = 10: cache read. If = 11: cache flush.

2.4 Address Spaces

The ST486DX/DX2 can directly address either memory or I/O space. Figure 2-22 illustrates the range of addresses available for memory address space and I/O address space. For the ST486DX/DX2, the addresses for physical memory range between 0000 0000h and FFFF FFFFh (4 GBytes). The accessible I/O addresses space ranges between 0000 0000h and 0000 FFFFh (64 KBytes). The ST486DX/DX2 does not use coprocessor communication space in upper I/O space between 8000 00F8h and 8000 00FFh as do the 386-style CPUs. The I/O locations 22h and 23h are used for ST486DX/DX2 configuration register access.

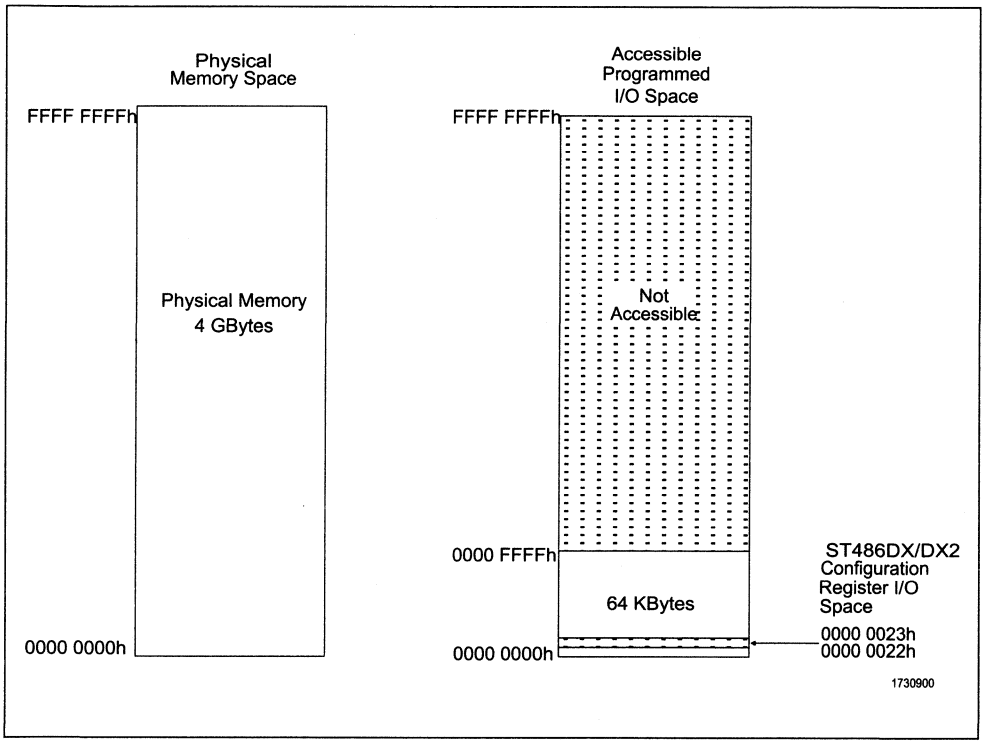


Figure 2 - 22. Memory and I/O Address Spaces

2.4.1 I/O Address Space

The ST486DX/DX2 I/O address space is accessed using IN and OUT instructions to addresses referred to as "ports". The accessible I/O address space is 64 KBytes and can be accessed as 8-bit, 16-bit or 32-bit ports. The execution of any IN or OUT instruction causes the M/IO# pin to be driven low, thereby selecting the I/O space instead of memory space.

The ST486DX/DX2 configuration registers reside within the I/O address space at port addresses 22h and 23h and are accessed using the standard IN and OUT instructions. The configuration registers are modified by writing the index of the configuration register to port 22h and then transferring the data through port 23h. Accesses to the on-chip configuration registers do not generate external I/O cycles. However, each port 23h operation must be preceded by a port 22h write with a valid index value. Otherwise, the second and later port 23h operations are directed off-chip and generate external I/O cycles without modifying the on-chip configuration registers. Also, writes to port 22h outside of the ST486DX/DX2 index range (C0h-CFh and FEh-FFh) result in external I/O cycles and do not effect the on-chip configuration registers. Reads of port 22h are always directed off-chip.

2.4.2 Memory Address Space

The ST486DX/DX2 directly addresses up to 4 GBytes of physical memory. Memory address space is accessed as bytes, words (16-bits) or doublewords (32-bits). Words and doublewords are stored in consecutive memory bytes with the low-order byte located in the lowest address. The physical address of a word or doubleword is the byte address of the low-order byte.

With the ST486DX/DX2, memory can be addressed using nine different addressing modes. These addressing modes are used to calculate an offset address often referred to as an effective address. Depending on the operating mode of the CPU, the offset is then combined using memory management mechanisms to create a physical address that actually addresses the physical memory devices.

Memory management mechanisms on the ST486DX/DX2 consist of segmentation and paging. Segmentation allows each program to use several independent, protected address spaces. Paging supports a memory subsystem that simulates a large address space using a small amount of RAM and disk storage for physical memory. Either or both of these mechanisms can be used for management of the ST486DX/DX2 memory address space.

2.4.2.1 Offset Mechanism

The offset mechanism computes an offset (effective) address by adding together up to three values: a base, an index and a displacement. The base, if present, is the value in one of eight 32-bit general registers at the time of the execution of the instruction. The index, like the base, is a value that is contained in one of the 32-bit general registers (except the ESP register) when the instruction is executed. The index differs from the base in that the index is first multiplied by a scale factor of 1, 2, 4 or 8 before the summation is made. The third component added to the memory address calcula-

tion is the displacement which is a value of up to 32-bits in length supplied as part of the instruction. Figure 2-23 illustrates the calculation of the offset address.

Nine valid combinations of the base, index, scale factor and displacement can be used with the ST486DX/DX2 instruction set. These combinations are listed in Table 2-19. The base and index both refer to contents of a register as indicated by [Base] and [Index].

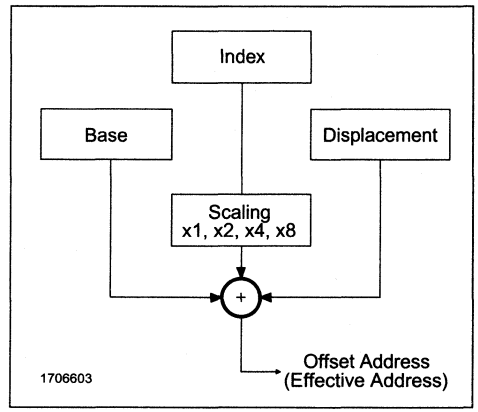


Figure 2 - 23. Offset Address Calculation

Table 2 - 19. Memory Addressing Modes

ADDRESSING MODE	BASE	INDEX	SCALE FACTOR (SF)	DISPLACEMENT (DP)	OFFSET ADDRESS (OA) CALCULATION
Direct				x	OA = DP
Register Indirect	x				OA = [BASE]
Based	x			x	OA = [BASE] + DP
Index		x		x	OA = [INDEX] + DP
Scaled Index		x	x	x	OA = ([INDEX] * SF) + DP
Based Index	x	x			OA = [BASE] + [INDEX]
Based Scaled Index	x	x	x		OA = [BASE] + ([INDEX] * SF)
Based Index with Displacement	x	x		x	OA = [BASE] + [INDEX] + DP
Based Scaled Index with Displacement	x	x	x	x	OA = [BASE] + ([INDEX] * SF) + DP

2.4.2.2 Real Mode Memory Addressing

In real mode operation, the ST486DX/DX2 only addresses the lowest 1 MByte of memory. To calculate a physical memory address, the 16-bit segment base address located in the selected segment register is multiplied by 16 and then the 16-bit offset address is added. The resulting 20-bit address is then extended with twelve zeros in the upper address bits to create the 32-bit physical address. Figure 2-24 illustrates the real mode address calculation.

The addition of the base address and the offset address may result in a carry. Therefore, the resulting address may actually contain up to 21 significant address bits that can address memory in the first 64 KBytes above 1 MByte.

2.4.2.3 Protected Mode Memory Addressing

In protected mode three mechanisms calculate a physical memory address (Figure 2-25, Page 58).

- **Offset Mechanism** that produces the offset or effective address as in real mode.
- **Selector Mechanism** that produces the base address.
- Optional **Paging Mechanism** that translates a linear address to the physical memory address.

The offset and base address are added together to produce the linear address. If paging is not used, the linear address is used as the physical memory address. If paging is enabled, the paging mechanism is used to translate the linear address into the physical address. The offset mechanism is described earlier in this section and applies to both real and protected mode. The selector and paging mechanisms are described in the following paragraphs.

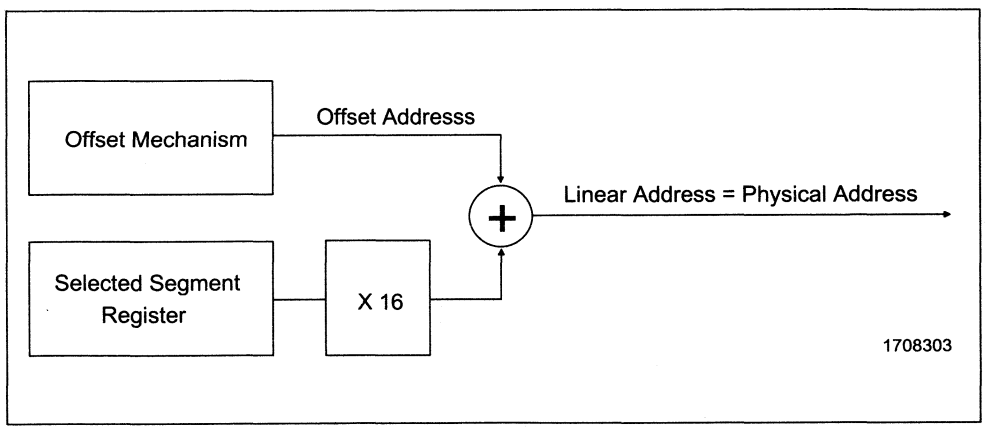


Figure 2 - 24. Real Mode Address Calculation

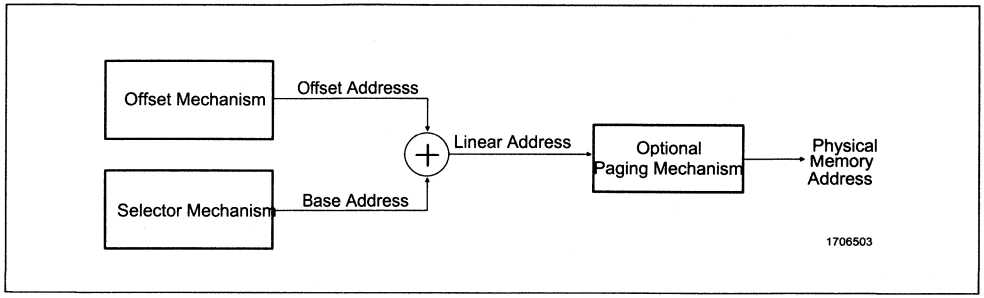


Figure 2-25. Protected Mode Address Calculation

Selector Mechanism

Memory is divided into an arbitrary number of segments, each containing usually much less than the 2^{32} byte (4 GByte) maximum.

The six segment registers (CS, DS, SS, ES, FS and GS) each contain a 16-bit selector that is used when the register is loaded to locate a segment descriptor in either the global descriptor table (GDT) or the local descriptor table (LDT). The segment descriptor defines the base address, limit and attributes of the selected segment and is cached on the ST486DX/DX2 as a result of loading the selector. The cached descriptor contents are not visible to the programmer. When a memory reference occurs in protected mode, the linear address is generated by adding the segment base address in the hidden portion of the segment register to the offset address. If paging is not enabled, this linear address is used as the physical memory address. Figure 2-26 illustrates the operation of the selector mechanism.

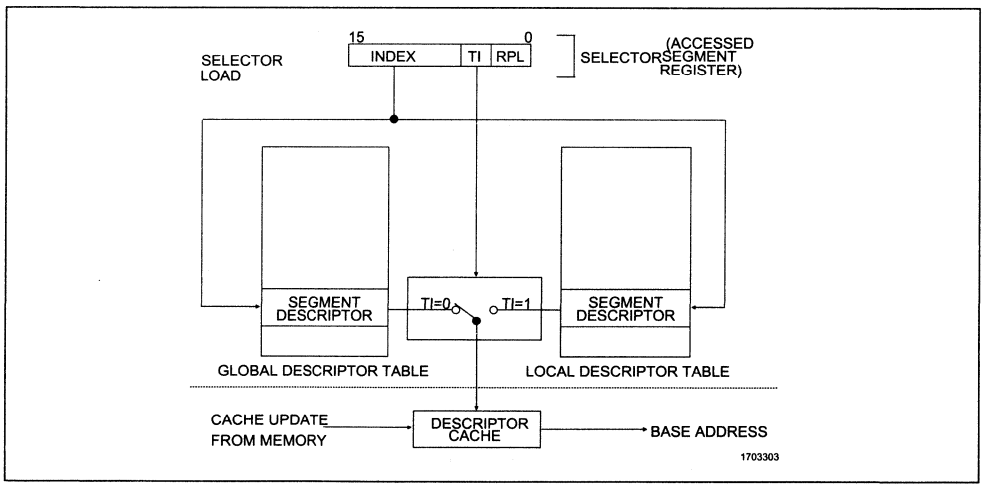


Figure 2-26. Selector Mechanism

Paging Mechanism

The paging mechanism supports a memory subsystem that simulates a large address space with a small amount of RAM and disk storage. The paging mechanism either translates a linear address to its corresponding physical address or generates an exception if the required page is not currently present in RAM. When the operating system services the exception, the required page is loaded into memory and the instruction is then restarted. Pages are either 4 KBytes or 1 MByte in size. The CPU defaults to 4 KByte pages that are aligned to 4 KByte boundaries.

A page is addressed by using two levels of tables as illustrated in Figure 2-27 (Page 60). The upper 10 bits of the 32-bit linear address are used to locate an entry in the **page directory table**. The page directory table acts as a 32-bit master index to up to 1K individual second-level page tables. The selected entry in the page directory table, referred to as the directory table entry, identifies the starting address of the second-level **page table**. The page directory table itself is a page and is, therefore, aligned to a 4 KByte boundary. The physical address of the current page directory table is stored in the CR3 control register, also referred to as the Page Directory Base Register (PDBR).

Bits 12-21 of the 32-bit linear address, referred to as the Page Table Index, locate a 32-bit entry in the second-level page table. This Page Table Entry (PTE) contains the base address of the desired page frame. The second-level page table addresses up to 1K individual page frames. A second-level page table is 4 KBytes in size and is itself a page. The lower 12 bits of the 32-bit linear address, referred to as the Page Frame Offset (PFO), locate the desired physical data within the page frame.

Since the page directory table can point to 1K page tables, and each page table can point to 1K of page frames, a total of 1M of page frames can be implemented. Since each page frame contains 4 KBytes, up to 4 GBytes of virtual memory can be addressed by the ST486DX/DX2 with a single page directory table.

In addition to the base address of the page table or the page frame, each directory table entry or page table entry contains attribute bits and a present bit as illustrated in Figure 2-28 (Page 60) and listed in Table 2-20 (Page 61).

If the present bit (P) is set in the DTE, the page table is present and the appropriate page table entry is read. If P=1 in the corresponding PTE (indicating that the page is in memory), the accessed and dirty bits are updated, if necessary, and the operand is fetched. Both accessed bits are set (DTE and PTE), if necessary, to indicate that the table and the page have been used to translate a linear address. The dirty bit (D) is set before the first write is made to a page.

The present bits must be set to validate the remaining bits in the DTE and PTE. If either of the present bits are not set, a page fault is generated when the DTE or PTE is accessed. If P=0, the remaining DTE/PTE bits are available for use by the operating system. For example, the operating system can use these bits to record where on the hard disk the pages are located. A page fault is also generated if the memory reference violates the page protection attributes.

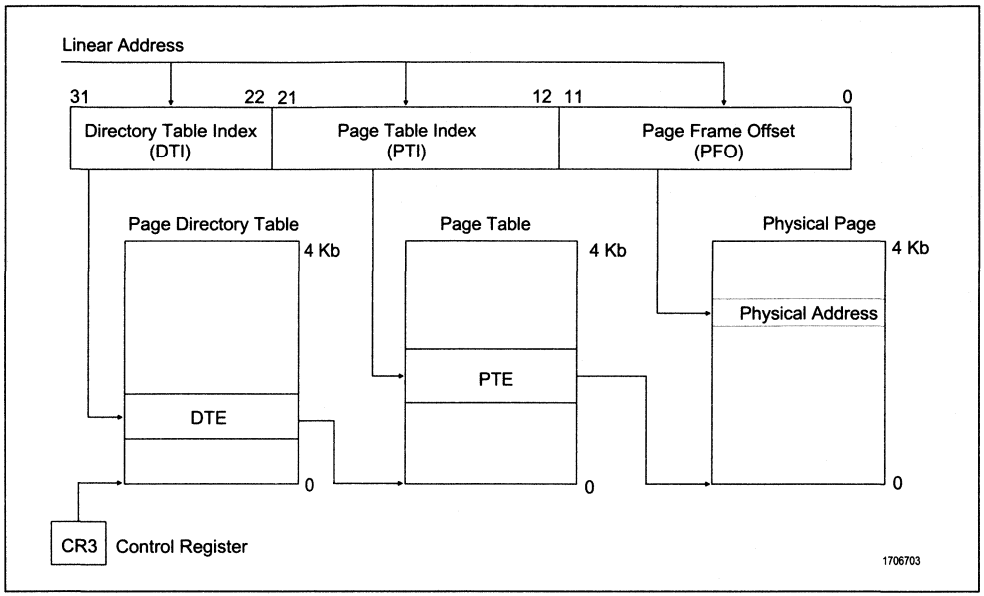


Figure 2 - 27. Paging Mechanism

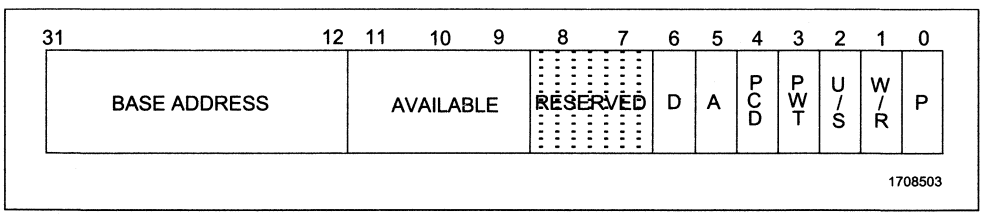


Figure 2 - 28. Directory and Page Table Entry (DTE and PTE) Format

Translation Look-Aside Buffer

The translation look-aside buffer (TLB) is a cache for the paging mechanism and replaces the two-level page table lookup procedure for TLB hits. The TLB is a four-way set associative 32-entry page table cache that automatically keeps the most commonly used page table entries in the processor. The 32-entry TLB, coupled with a 4K page size, results in coverage of 128 KBytes of memory addresses.

The TLB must be flushed when entries in the page tables are changed. The TLB is flushed whenever the CR3 register is loaded. An individual entry in the TLB can be flushed using the INVLPG instruction.

Table 2 - 20. Directory and Page Table Entry (DTE and PTE) Bit Definitions

BIT POSITION	FIELD NAME	DESCRIPTION
31-12	BASE ADDRESS	Specifies the base address of the page or page table.
11-9	--	Undefined and available to the programmer.
8-7	--	Reserved and not available to the programmer.
6	D	Dirty Bit. If set, indicates that a write access has occurred to the page (PTE only, undefined in DTE).
5	A	Accessed Flag. If set, indicates that a read access or write access has occurred to the page.
4	PCD	Page Caching Disable Flag. If set, indicates that the page is not cacheable in the on-chip cache.
3	PWT	Page Write-Through Flag. If set, indicates that writes to the page or page tables that hit in the on-chip cache must update both the cache and external memory.
2	U/S	User/Supervisor Attribute. If set (user), page is accessible at privilege level 3. If clear (supervisor), page is accessible only when CPL ≤ 2.
1	W/R	Write/Read Attribute. If set (write), page is writable. If clear (read), page is read only.
0	P	Present Flag. If set, indicates that the page is present in RAM memory, and validates the remaining DTE/PTE bits. If clear, indicates that the page is not present in memory and the remaining DTE/PTE bits can be used by the programmer.

2.5 Interrupts and Exceptions

The processing of either an interrupt or an exception changes the normal sequential flow of a program by transferring program control to a selected service routine. Except for SMM interrupts, the location of the selected service routine is determined by one of the interrupt vectors stored in the interrupt descriptor table.

True interrupts are hardware interrupts and are generated by signal sources external to the CPU. All exceptions (including so-called software interrupts) are produced internally by the CPU.

2.5.1 Interrupts

External events can interrupt normal program execution by using one of the three interrupt pins on the ST486DX/DX2.

- Non-maskable Interrupt (NMI pin)
- Maskable Interrupt (INTR pin)
- SMM Interrupt (SMI# pin).

For most interrupts, program transfer to the interrupt routine occurs after the current instruction has been completed. When the execution returns to the original program, it begins immediately following the interrupted instruction.

The **NMI interrupt** cannot be masked by software and always uses interrupt vector 2 to locate its service routine. Since the interrupt vector is fixed and is supplied internally, no interrupt acknowledge bus cycles are performed. This interrupt is normally reserved for unusual situations such as parity errors and has priority over INTR interrupts.

Once NMI processing has started, no additional NMIs are processed until an IRET instruction is executed, typically at the end of the NMI service routine. If NMI is re-asserted prior to execution of the IRET instruction, one and only one NMI rising edge is stored and then processed after execution of the next IRET.

During the NMI service routine, maskable interrupts may be enabled. If an unmasked INTR occurs during the NMI service routine, the INTR is serviced and execution returns to the NMI service routine following the next IRET. If a HALT instruction is executed within the NMI service routine, the ST486DX/DX2 restarts execution only in response to RESET, an unmasked INTR or an SMM interrupt. NMI does not restart CPU execution under this condition.

The **INTR interrupt** is unmasked when the Interrupt Enable Flag (IF) in the EFLAGS register is set to 1. With the exception of string operations, INTR interrupts are acknowledged between instructions. Long string operations have interrupt windows between memory moves that allow INTR interrupts to be acknowledged.

When an INTR interrupt occurs, the CPU performs two locked interrupt acknowledge bus cycles. During the second cycle, the CPU reads an 8-bit vector which is supplied by an external interrupt controller. This vector selects which of the 256 possible interrupt handlers will be executed in response to the interrupt.

The **SMM interrupt** has higher priority than either INTR or NMI. After SMI# is asserted, program execution is passed to an SMI service routine which runs in SMM address space reserved for this purpose. The remainder of this section does not apply to the SMM interrupts. SMM interrupts are described in greater detail later in this chapter.

2.5.2 Exceptions

Exceptions are generated by an interrupt instruction or a program error. Exceptions are classified as traps, faults or aborts depending on the mechanism used to report them and the restartability of the instruction which first caused the exception.

A **Trap Exception** is reported immediately following the instruction that generated the trap exception. Trap exceptions are generated by execution of a software interrupt instruction (INTO, INT 3, INT n, BOUND), by a single-step operation or by a data breakpoint.

Software interrupts can be used to simulate hardware interrupts. For example, an INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table. Execution of the interrupt service routine occurs regardless of the state of the IF flag in the EFLAGS register.

The one byte INT 3, or breakpoint interrupt (vector 3), is a particular case of the INT n instruction. By inserting this one byte instruction in a program, the user can set breakpoints in the code that can be used during debug.

Single-step operation is enabled by setting the TF bit in the EFLAGS register. When TF is set, the CPU generates a debug exception (vector 1) after the execution of every instruction. Data breakpoints also generate a debug exception and are specified by loading the debug registers (DR0-DR7) with the appropriate values.

A **Fault Exception** is caused by a program error and is reported prior to completion of the instruction that generated the exception. By reporting the fault prior to instruction completion, the CPU is left in a state which allows the instruction to be restarted and the effects of the faulting instruction to be nullified. Fault exceptions include divide-by-zero errors, invalid opcodes, page faults and coprocessor errors. Debug exceptions (vector 1) are also handled as faults (except for data breakpoints and single-step operations). After execution of the fault service routine, the instruction pointer points to the instruction that caused the fault.

An **Abort Exception** is a type of fault exception that is severe enough that the CPU cannot restart the program at the faulting instruction. The double fault (vector 8) is the only abort exception that occurs on the ST486DX/DX2.

2.5.3 Interrupt Vectors

When the CPU services an interrupt or exception, the current program's instruction pointer and flags are pushed onto the stack to allow resumption of execution of the interrupted program. In protected mode, the processor also saves an error code for some exceptions. Program control is then transferred to the interrupt handler (also called the interrupt service routine). Upon execution of an IRET at the end of the service routine, program execution resumes at the instruction pointer address saved on the stack when the interrupt was serviced.

Interrupt Vector Assignments

Each interrupt (except SMI#) and exception is assigned one of 256 interrupt vector numbers Table 2-21. The first 32 interrupt vector assignments are defined or reserved. INT instructions acting as software interrupts may use any of interrupt vectors, 0 through 255.

The non-maskable hardware interrupt (NMI) is the assigned vector 2. Illegal opcodes including faulty FPU instructions will cause an illegal opcode exception, interrupt vector 6.

Table 2 - 21. Interrupt Vector Assignments

INTERRUPT VECTOR	FUNCTION	EXCEPTION TYPE
0	Divide error	FAULT
1	Debug exception	TRAP/FAULT*
2	NMI interrupt	
3	Breakpoint	TRAP
4	Interrupt on overflow	TRAP
5	BOUND range exceeded	FAULT
6	Invalid opcode	FAULT
7	Device not available	FAULT
8	Double fault	ABORT
9	Reserved	
10	Invalid TSS	FAULT
11	Segment not present	FAULT
12	Stack fault	FAULT
13	General protection fault	TRAP/FAULT
14	Page fault	FAULT
15	Reserved	
16	FPU error	FAULT
17	Alignment check exception	FAULT
18-31	Reserved	
32-255	Maskable hardware interrupts	TRAP
0-255	Programmed interrupt	TRAP
*Note: Data breakpoints and single-steps are traps. All other debug exceptions are faults.		

In response to a maskable hardware interrupt (INTR), the ST486DX/DX2 issues interrupt acknowledge bus cycles used to read the vector number from external hardware. These vectors should be in the range 32 - 255 as vectors 0 - 31 are pre-defined.

Interrupt Descriptor Table

The interrupt vector number is used by the ST486DX/DX2 to locate an entry in the interrupt descriptor table (IDT). In real mode, each IDT entry consists of a four-byte far pointer to the beginning of the corresponding interrupt service routine. In protected mode, each IDT entry is an eight-byte descriptor. The Interrupt Descriptor Table Register (IDTR) specifies the beginning address and limit of the IDT. Following reset, the IDTR contains a base address of 0h with a limit of 3FFh.

The IDT can be located anywhere in physical memory as determined by the IDTR register. The IDT may contain different types of descriptors: interrupt gates, trap gates and task gates. Interrupt gates are used primarily to enter a hardware interrupt handler. Trap gates are generally used to enter an exception handler or software interrupt handler. If an interrupt gate is used, the Interrupt Enable Flag (IF) in the EFLAGS register is cleared before the interrupt handler is entered. Task gates are used to make the transition to a new task.

2.5.4 Interrupt and Exception Priorities

As the ST486DX/DX2 executes instructions, it follows a consistent policy for prioritizing exceptions and hardware interrupts. The priorities for competing interrupts and exceptions are listed in Table 2-22 (Page 66). SMM interrupts always take precedence. Debug traps for the previous instruction and next instructions are handled as the next priority. When NMI and maskable INTR interrupts are both detected at the same instruction boundary, the ST486DX/DX2 microprocessor services the NMI interrupt first.

The ST486DX/DX2 checks for exceptions in parallel with instruction decoding and execution. Several exceptions can result from a single instruction. However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should make the appropriate corrections to the instruction and then restart the instruction. In this way, exceptions can be serviced until the instruction executes properly.

The ST486DX/DX2 supports instruction restart after all faults, except when an instruction causes a task switch to a task whose task state segment (TSS) is partially not present. A TSS can be partially not present if the TSS is not page aligned and one of the pages where the TSS resides is not currently in memory.

Table 2 - 22. Interrupt and Exception Priorities

PRIORITY	DESCRIPTION	NOTES
0	SMM hardware interrupt.	SMM interrupts are caused by SMI# asserted and always have highest priority.
1	Debug traps and faults from previous instruction.	Includes single-step trap and data breakpoints specified in the debug registers.
2	Debug traps for next instruction.	Includes instruction execution breakpoints specified in the debug registers.
3	Non-maskable hardware interrupt.	Caused by NMI asserted.
4	Maskable hardware interrupt.	Caused by INTR asserted and IF = 1.
5	Faults resulting from fetching the next instruction.	Includes segment not present, general protection fault and page fault.
6	Faults resulting from instruction decoding.	Includes illegal opcode, instruction too long, or privilege violation.
7	WAIT instruction and TS = 1 and MP = 1.	Device not available exception generated.
8	ESC instruction and EM = 1 or TS = 1.	Device not available exception generated.
9	Floating point error exception.	Caused by unmasked floating point exception with NE = 1.
10	Segmentation faults (for each memory reference required by the instruction) that prevent transferring the entire memory operand.	Includes segment not present, stack fault, and general protection fault.
11	Page Faults that prevent transferring the entire memory operand.	
12	Alignment check fault.	

2.5.5 Exceptions in Real Mode

Many of the exceptions described in Table 2-22 (Page 66) are not applicable in real mode. Exceptions 10, 11, and 14 do not occur in real mode. Other exceptions have slightly different meanings in real mode as listed in Table 2-23.

Table 2 - 23. Exception Changes in Real Mode

VECTOR NUMBER	PROTECTED MODE FUNCTION	REAL MODE FUNCTION
8	Double fault.	Interrupt table limit overrun.
10	Invalid TSS.	--
11	Segment not present.	--
12	Stack fault.	SS segment limit overrun.
13	General protection fault.	CS, DS, ES, FS, GS segment limit overrun.
14	Page fault.	--
Note: -- = does not occur		

2.5.6 Error Codes

When operating in protected mode, the following exceptions generate a 16-bit error code:

Double Fault
Alignment Check
Invalid TSS
Segment Not Present
Stack Fault
General Protection Fault
Page Fault.

The error code format is shown in Figure 2-29 and the error code bit definitions are listed in Table 2-24. Bits 15-3 (selector index) are not meaningful if the error code was generated as the result of a page fault. The error code is always zero for double faults and alignment check exceptions.

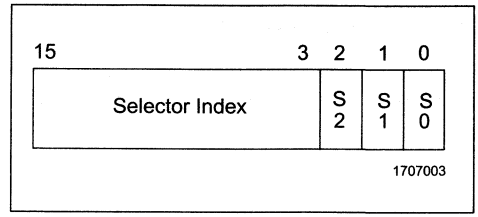


Figure 2 - 29. Error Code Format

Table 2 - 24. Error Code Bit Definitions

FAULT TYPE	SELECTOR INDEX (BITS 15-3)	S2 (BIT 2)	S1 (BIT 1)	S0 (BIT 0)
Page Fault	Reserved.	Fault caused by: 0 = not present page 1 = page-level protection violation.	Fault occurred during: 0 = read access 1 = write access.	Fault occurred during: 0 = supervisor access 1 = user access.
IDT Fault	Index of faulty IDT selector.	Reserved.	1	If = 1, exception occurred while trying to invoke exception or hardware interrupt handler.
Segment Fault	Index of faulty selector.	TI bit of faulty selector.	0	If =1, exception occurred while trying to invoke exception or hardware interrupt handler.

2.6 System Management Mode

System Management Mode (SMM) provides an additional interrupt which can be used for system power management or software transparent emulation of I/O peripherals. SMM is entered using the System Management Interrupt (SMI#) that has a higher priority than any other interrupt, including NMI. An SMI interrupt can also be triggered via the software using an SMINT instruction. After an SMI interrupt, portions of the CPU state are automatically saved, SMM is entered, and program execution begins at the base of SMM address space (Figure 2-30). Running in protected SMM address space, the interrupt routine does not interfere with the operating system or any application program.

Eight SMM instructions have been added to the 486DX instruction set that permit software initiated SMM, and saving and restoring of the total CPU state when in SMM mode. Two new pins, SMI# and SMADS#, support SMM functions.

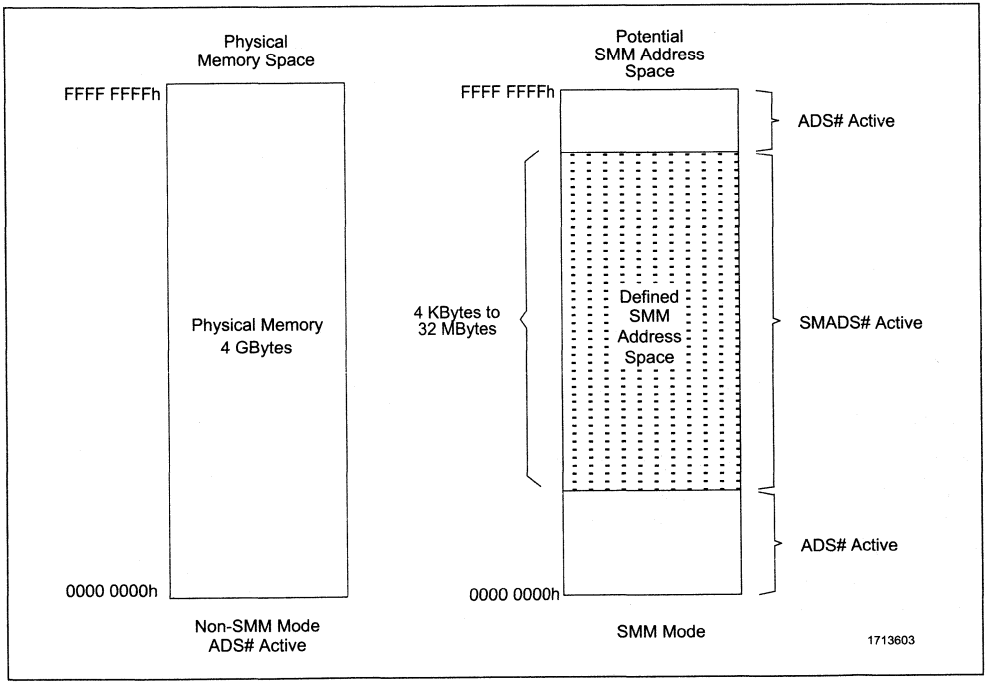


Figure 2 - 30. System Management Memory Address Space

2.6.1 SMM Operation

SMM operation is summarized in Figure 2-31. Entering SMM requires the assertion of the SMI# pin for at least two CLK periods or execution of the SMINT instruction. For the SMI# or SMINT instruction to be recognized, the following configuration register bits must be set as shown in Table 2-25. The configuration registers are discussed in detail earlier in this chapter.

Table 2 - 25. Requirement for Recognizing SMI# and SMINT

REGISTER (Bit)		SMI#	SMINT
SMI	CCR1 (1)	1	1
SMAC	CCR1 (2)	0	1
SMAR	SIZE (3-0)	>0	>0

After recognizing SMI# or SMINT and prior to executing the SMI service routine, some of the CPU state information is changed. Prior to modification, this information is automatically saved in the SMM memory space header located at the top of SMM memory space. After the header is saved, the CPU enters real mode and begins executing the SMI service routine starting at the SMM memory base address.

The SMI service routine is user definable and may contain system or power management software. If the power management software forces the CPU to power down, or the SMI service routine modifies more than what is automatically saved, the complete CPU state information can be saved.

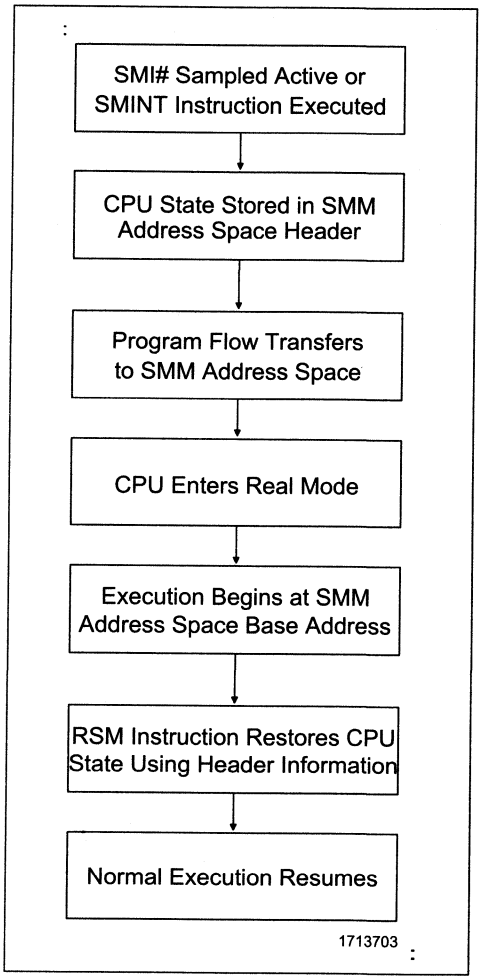


Figure 2 - 31. SMI Execution Flow Diagram

2.6.2 SMM Memory Space Header

With every SMI interrupt or SMINT instruction, certain CPU state information is automatically saved in the SMM memory space header located at the top of SMM address space (Figure 2-32 and Table 2-26 (Page 72).

The header contains CPU state information that is modified when servicing an SMI interrupt. Included in this information are two pointers. The Current IP points to the instruction executing when the SMI was detected.

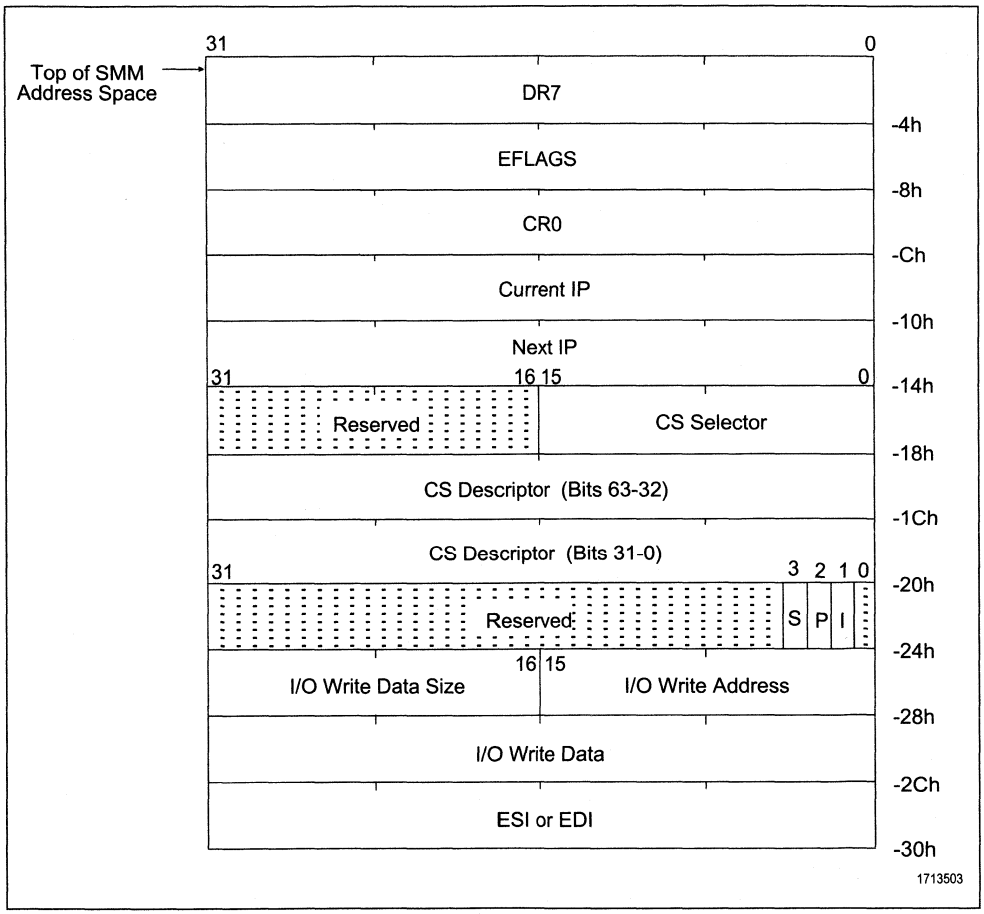


Figure 2 - 32. SMM Memory Space Header

The Next IP points to the instruction that will be executed after exiting SMM. Also saved are the contents of debug register 7 (DR7), the extended flags register (EFLAGS), and control register 0 (CR0). If SMM has been entered due to an I/O trap for a REP INSx or REP OUTSx instruction, the Current IP and Next IP fields contain the same addresses and the I and P field contain valid information.

If entry into SMM was caused by an I/O trap (see I/O Trapping, page 113), it is useful for the programmer to know the port address, data size and data value associated with that I/O operation. This information is also saved in the header and is only valid for an I/O write operation. The I/O write information is not restored within the CPU when executing a RSM instruction.

Table 2 - 26. SMM Memory Space Header

NAME	DESCRIPTION	SIZE
DR7	The contents of Debug Register 7.	4 Bytes
EFLAGS	The contents of Extended Flags Register.	4 Bytes
CR0	The contents of Control Register 0.	4 Bytes
Current IP	The address of the instruction executed prior to servicing SMI interrupt.	4 Bytes
Next IP	The address of the next instruction that will be executed after exiting SMM mode.	4 Bytes
CS Selector	Code segment register selector for the current code segment.	2 Bytes
CS Descriptor	Code segment register descriptor for the current code segment.	8 Bytes
S	Software SMM Entry Indicator. S = 1, if current SMM is the result of an SMINT instruction. S = 0, if current SMM is not the result of an SMINT instruction.	1 Bit
P	REP INSx/OUTSx Indicator. P = 1 if current instruction has a REP prefix. P = 0 if current instruction does not have a REP prefix.	1 Bit
I	IN, INSx, OUT, or OUTSx Indicator. I = 1 if current instruction performed is an I/O WRITE. I = 0 if current instruction performed is an I/O READ.	1 Bit
I/O Write Data Size	Indicates size of data for the trapped I/O write. 01h = byte 03h = word 0Fh = dword	2 Bytes
I/O Write Address	Address of the trapped I/O write.	2 Bytes
I/O Write Data	Data associated with the trapped I/O write.	4 Bytes
ESI or EDI	Restored ESI or EDI value. Used when it is necessary to repeat a REP OUTSx or REP INSx instruction when one of the I/O cycles caused an SMI# trap.	4 Bytes
Note: INSx = INS, INSB, INSW or INSD instruction. Note: OUTSx = OUTS, OUTSB, OUTSW and OUTSD instruction.		

2.6.3 SMM Instructions

The ST486DX/DX2 automatically saves the minimal amount of CPU state information when entering SMM which allows fast SMI service routine entry and exit. After entering the SMI service routine, the MOV, SVDC, SVLDT and SVTS instructions can be used to save the complete CPU state information. If the SMI service routine modifies more than what is automatically saved or forces the CPU to power down, the complete CPU state information must be saved. Since the CPU is a static device, its internal state is retained when the input clock is stopped. Therefore, an entire CPU state save is not necessary prior to stopping the input clock.

The new SMM instructions, listed in Table 2-27, can only be executed if:

SMI# is enabled **and**
SMAR SIZE 0 **and**
[the Current Privilege Level (CPL) = 0 and The SMAC bit (CCR1, bit 2) is set] **or**
[the Current Privilege Level (CPL) = 0 **and** the CPU is in an SMI service routine (SMI# = 0)].

If the above conditions are not met and an attempt is made to execute an SVDC, RSDC, SVLDT, RSLDT, SVTS, RSTS, SMINT or RSM instruction, an invalid opcode exception is generated. These instructions can be executed outside of defined SMM space provided the above conditions are met.

Table 2- 27. SMM Instruction Set

INSTRUCTION	OPCODE	FORMAT	DESCRIPTION
SVDC	0F 78 [mod sreg3 r/m]	SVDC mem80, sreg3	*Save Segment Register and Descriptor* Saves reg (DS, ES, FS, GS, or SS) to mem80.
RSDC	0F 79 [mod sreg3 r/m]	RSDC sreg3, mem80	*Restore Segment Register and Descriptor* Restores reg (DS, ES, FS, GS, or SS) from mem80. Use RSM to restore CS. Note: Processing "RSDC CS, Mem80" will produce an exception.
SVLDT	0F 7A [mod 000 r/m]	SVLDT mem80	*Save LDTR and Descriptor* Saves Local Descriptor Table (LDTR) to mem80.
RSLDT	0F 7B [mod 000 r/m]	RSLDT mem80	*Restore LDTR and Descriptor* Restores Local Descriptor Table (LDTR) from mem80.
SVTS	0F 7C [mod 000 r/m]	SVTS mem80	*Save TSR and Descriptor* Saves Task State Register (TSR) to mem80.
RSTS	0F 7D [mod 000 r/m]	RSTS mem80	*Restore TSR and Descriptor* Restores Task State Register (TSR) from mem80.
SMINT	0F 7E	SMINT	*Software SMM Entry* CPU enters SMM mode. CPU state information is saved in SMM memory space header and execution begins at SMM base address.
RSM	0F AA	RSM	*Resume Normal Mode* Exits SMM mode. The CPU state is restored using the SMM memory space header and execution resumes at interrupted point.
Note: mem80 = 80-bit memory location			

The SMINT instruction can be used by software to enter SMM. The CPU will not drive the SMI# output low during the software initiated SMM.

However, if the SMI# is asserted to the CPU during a software SMM, the SMI# handshake occurs normally. The hardware SMI# is serviced after the software SMM has been exited by execution of the RSM instruction.

All of the SMM instructions (except RSM and SMINT) save or restore 80 bits of data, allowing the saved values to include the hidden portion of the register contents.

2.6.4 SMM Memory Space

SMM memory space is defined by specifying the base address and size of the SMM memory space in the SMAR register. The base address must be a multiple of the SMM memory space size. For example, a 32 KByte SMM memory space must be located at a 32 KByte address boundary. The memory space size can range from 4 KBytes to 32 MBytes.

SMM memory space accesses are always non-cacheable. SMM accesses ignore the state of the A20M# input pin and drive the A20 address bit to the unmasked value.

Access to the SMM memory space can be made while not in SMM mode by setting the SMAC bit in the CCR1 register. This feature may be used to initialize the SMM memory space.

While in SMM mode, SMADS# address strobes are generated instead of ADS# for SMM memory accesses. Any memory accesses outside the defined SMM space result in normal memory accesses and ADS# strobes. Data (non-code) accesses to main memory that overlap with defined SMM memory space are allowed if MMAC in CCR1 is set. In this case, ADS# strobes are generated for data accesses only and SMADS# strobes continue to be generated for code accesses.

2.6.5 SMI Service Routine Execution

Upon entry into SMM, after the SMM header has been saved, the CR0, EFLAGS, and DR7 registers are set to their reset values. The Code Segment (CS) register is loaded with the base, as defined by the SMAR register, and a limit of 4 GBytes. The SMI service routine then begins execution at the SMM base address in real mode.

The programmer must save the value of any registers that may be changed by the SMI service routine. For data accesses immediately after entering the SMI service routine, the programmer must use CS as a segment override. I/O port access is possible during the routine but care must be taken to save registers modified by the I/O instructions. Before using a segment register, the register and the register's descriptor cache contents should be saved using the SVDC instruction. While executing in the SMM space, execution flow can transfer to normal memory locations.

Hardware interrupts, (INTRs and NMIs), may be serviced during a SMI service routine. If interrupts are to be serviced while executing in the SMM memory space, the SMM memory space must be within the 0 to 1 MByte address range to guarantee proper return to the SMI service routine after handling the interrupt.

INTRs are automatically disabled when entering SMM since the IF flag is set to its reset value. Once in SMM, the INTR can be enabled by setting the IF flag. An NMI event in SMM mode can be enabled by setting NMIEN

in the CCR3 register. If NMI is not enabled while in SMM mode, the CPU latches one NMI event and services the interrupt after NMI has been enabled or after exiting SMM mode through the RSM instruction.

Within the SMI service routine, protected mode may be entered and exited as required, and real or protected mode device drivers may be called.

To exit the SMI service routine, a Resume (RSM) instruction, rather than an IRET, is executed. The RSM instruction causes the ST486DX/DX2 to restore the CPU state using the SMM header information and resume execution at the interrupted point. If the full CPU state was saved by the programmer, the stored values should be reloaded prior to executing the RSM instruction using the MOV, RSDC, RSLDT and RSTS instructions.

CPU States Related to SMM and Suspend Mode

The state diagram shown in Figure 2-33 (Page 76) illustrates the various CPU states associated with SMM and suspend mode. While in the SMI service routine, the ST486DX/DX2 can enter suspend mode either by (1) executing a halt (HLT) instruction or (2) by asserting the SUSP# input.

During SMM operations and while in SUSP# initiated suspend mode, an occurrence of either NMI or INTR is latched. (In order for INTR to be latched, the IF flag must be set.) The INTR or NMI is serviced after exiting suspend mode.

If suspend mode is entered via a HLT instruction from the operating system or application software, the reception of an SMI# interrupt causes the CPU to exit suspend mode and enter SMM. If suspend mode is entered via the hardware (SUSP# = 0) while the operating system or application software is active, the CPU latches one occurrence of INTR, NMI and SMI#.

2.7 Shutdown and Halt

The **Halt Instruction** (HLT) stops program execution and prevents the processor from using the local bus until restarted. The ST486DX/DX2 then enters a low-power suspend mode if the HLT bit in CCR2 is set. SMI, NMI, INTR with interrupts enabled (IF bit in EFLAGS=1), or RESET forces the CPU out of the halt state. If interrupted, the saved code segment and instruction pointer specify the instruction following the HLT.

Shutdown occurs when a severe error is detected that prevents further processing. An NMI input can bring the processor out of shutdown if the IDT limit is large enough to contain the NMI interrupt vector (at least 000Fh) and the stack has enough room to contain the vector and flag information (i.e., stack pointer is greater than 0005h). Otherwise, shutdown can only be exited by a processor reset.

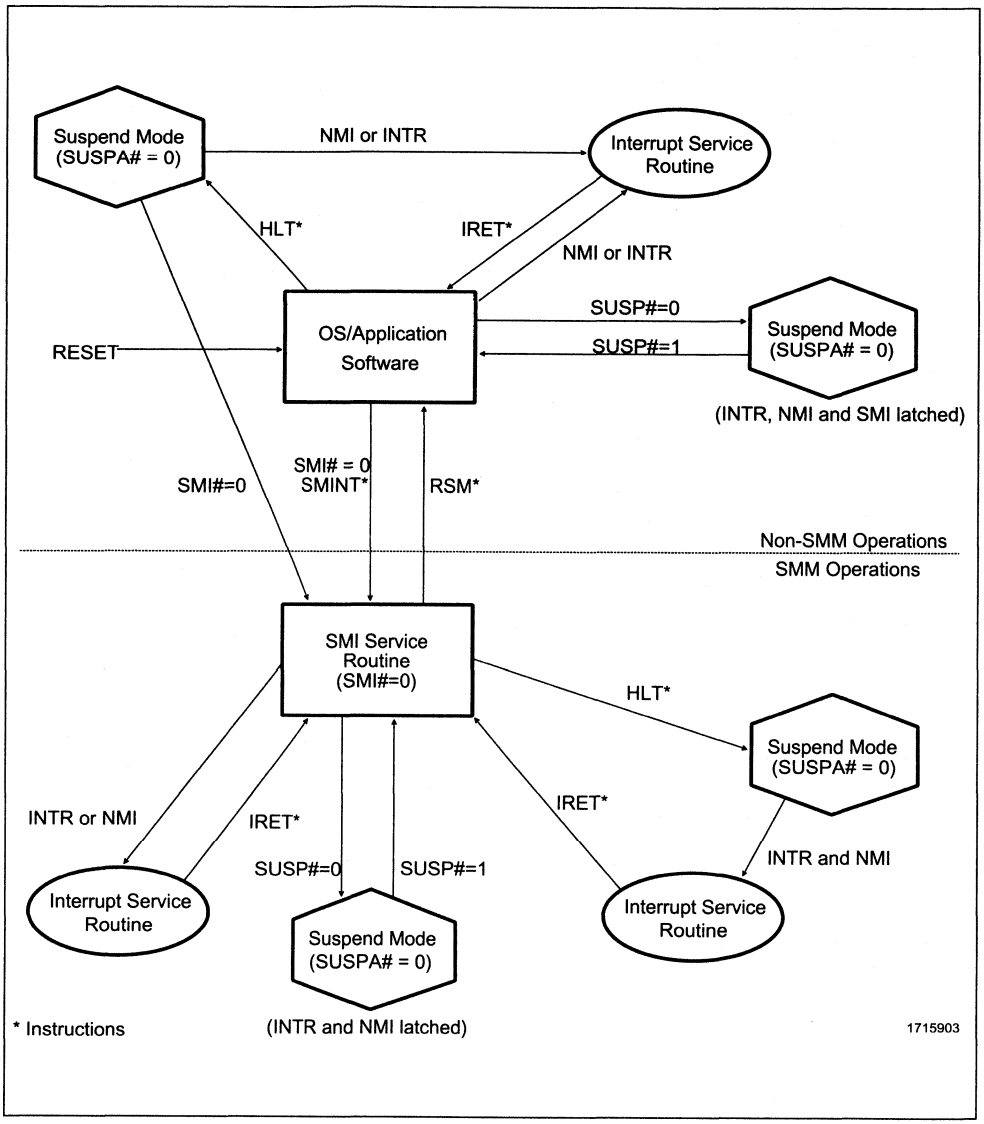


Figure 2 - 33. SMM and Suspend Mode State Diagram

2.8 Protection

Segment protection and page protection are safeguards built into the ST486DX/DX2 protected mode architecture which deny unauthorized or incorrect access to selected memory addresses. These safeguards allow multitasking programs to be isolated from each other and from the operating system. Page protection is discussed earlier in this chapter in Section 2.4. This section concentrates on segment protection.

Selectors and descriptors are the key elements in the segment protection mechanism. The segment base address, size, and privilege level are established by a segment descriptor. Privilege levels control the use of privileged instructions, I/O instructions and access to segments and segment descriptors. Selectors are used to locate segment descriptors.

Segment accesses are divided into two basic types, those involving code segments (e.g., control transfers) and those involving data accesses. The ability of a task to access a segment depends on:

- the segment type
- the instruction requesting access
- the type of descriptor used to define the segment
- the associated privilege levels (described below).

Data stored in a segment can be accessed only by code executing at the same or a more privileged level. A code segment or procedure can only be called by a task executing at the same or a less privileged level.

2.8.1 Privilege Levels

The values for privilege levels range between 0 and 3. Level 0 is the highest privilege level (most privileged), and level 3 is the lowest privilege level (least privileged). The privilege level in real mode is effectively 0.

The **Descriptor Privilege Level** (DPL) is the privilege level defined for a segment in the segment descriptor. The DPL field specifies the minimum privilege level needed to access the memory segment pointed to by the descriptor.

The **Current Privilege Level** (CPL) is defined as the current task's privilege level. The CPL of an executing task is stored in the hidden portion of the code segment register and essentially is the DPL for the current code segment.

The **Requested Privilege Level** (RPL) specifies a selector's privilege level and is used to distinguish between the privilege level of a routine actually accessing memory (the CPL), and the privilege level of the original requestor (the RPL) of the memory access. The lesser of the RPL and CPL is called the effective privilege level (EPL). Therefore, if RPL = 0 in a segment selector, the effective privilege level is always determined by the CPL. If RPL = 3, the effective privilege level is always 3 regardless of the CPL.

For a memory access to succeed, the effective privilege level (EPL) must be at least as privileged as the descriptor privilege level (EPL $\leq$ DPL). If the EPL is less privileged than the DPL (EPL>DPL), a general protection fault is generated. For example, if a segment has a DPL = 2, an instruction accessing the segment only succeeds if executed with an EPL $\leq$ 2.

2.8.2 I/O Privilege Levels

The I/O Privilege Level (IOPL) allows the operating system executing at CPL=0 to define the least privileged level at which IOPL-sensitive instructions can unconditionally be used. The IOPL-sensitive instructions include CLI, IN, OUT, INS, OUTS, REP INS, REP OUTS, and STI. Modification of the IF bit in the EFLAGS register is also sensitive to the I/O privilege level.

The IOPL is stored in the EFLAGS register. An I/O permission bit map is available as defined by the 32-bit Task State Segment (TSS). Since each task can have its own TSS, access to individual I/O ports can be granted through separate I/O permission bit maps.

If CPL $\leq$ IOPL, IOPL-sensitive operations can be performed. If CPL > IOPL, a general protection fault is generated if the current task is associated with a 16-bit TSS. If the current task is associated with a 32-bit TSS and CPL > IOPL, the CPU consults the I/O permission bitmap in the TSS to determine on a port-by-port basis whether or not I/O instructions (IN, OUT, INS, OUTS, REP INS, REP OUTS) are permitted, and the remaining IOPL-sensitive operations generate a general protection fault.

2.8.3 Privilege Level Transfers

A task's CPL can be changed only through intersegment control transfers using gates or task switches to a code segment with a different privilege level. Control transfers result from exception and interrupt servicing and from execution of the CALL, JMP, INT, IRET and RET instructions.

There are five types of control transfers that are summarized in Table 2-28 (Page 79). Control transfers can be made only when the operation causing the control transfer references the correct descriptor type. Any violation of these descriptor usage rules causes a general protection fault.

Any control transfer that changes the CPL within a task results in a change of stack. The initial values for the stack segment (SS) and stack pointer (ESP) for privilege levels 0, 1, and 2 are stored in the TSS. During a JMP or CALL control transfer, the SS and ESP are loaded with the new stack pointer and the previous stack pointer is saved on the new stack. When returning to the original privilege level, the RET or IRET instruction restores the less-privileged stack.

Table 2-28. Descriptor Types Used for Control Transfer

TYPE OF CONTROL TRANSFER	OPERATION TYPES	DESCRIPTOR REFERENCED	DESCRIPTOR TABLE
Intersegment within the same privilege level.	JMP, CALL, RET, IRET*	Code Segment	GDT or LDT
Intersegment to the same or a more privileged level. Interrupt within task (could change CPL level).	CALL	Gate Call	GDT or LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	LDT
Intersegment to a less privileged level (changes task CPL).	RET, IRET*	Code Segment	GDT or LDT
Task Switch via TSS	CALL, JMP	Task State Segment	GDT.
Task Switch via Task Gate	CALL, JMP	Task Gate	GDT or LDT
	IRET**, Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT
* NT (Nested Task bit in EFLAGS) = 0 ** NT (Nested Task bit in EFLAGS) = 1			

2.8.3.1 Gates

Gate descriptors provide protection for privilege transfers among executable segments. Gates are used to transition to routines of the same or a more privileged level. Call gates, interrupt gates and trap gates are used for privilege transfers within a task. Task gates are used to transfer between tasks.

Gates conform to the standard rules of privilege. In other words, gates can be accessed by a task if the effective privilege level (EPL) is the same or more privileged than the gate descriptor's privilege level (DPL).

2.8.4 Initialization and Transition to Protected Mode

The ST486DX/DX2 microprocessor switches to real mode immediately after RESET. While operating in real mode, the system tables and registers should be initialized. The GDTR and IDTR must point to a valid GDT and IDT, respectively. The size of the IDT should be at least 256 bytes, and the GDT must contain descriptors which describe the initial code and data segments.

The processor can be placed in protected mode by setting the PE bit in the CR0 register. After enabling protected mode, the CS register should be loaded and the instruction decode queue should be flushed by executing an intersegment JMP. Finally, all data segment registers should be initialized with appropriate selector values.

2.9 Virtual 8086 Mode

Both real mode and virtual 8086 (V86) mode are supported by the ST486DX/DX2 CPU allowing execution of 8086 application programs and 8086 operating systems. V86 Mode allows the execution of 8086-type applications, yet still permits use of the ST486DX/DX2 protection mechanism. V86 tasks run at privilege level 3. Upon entry, all segment limits are set to FFFFh (64K) as in real mode.

2.9.1 Memory Addressing

While in V86 mode, segment registers are used in an identical fashion to real mode. The contents of the segment register are multiplied by 16 and added to the offset to form the segment base linear address. The ST486DX/DX2 CPU permits the operating system to select which programs use the V86 address mechanism and which programs use protected mode addressing for each task.

The ST486DX/DX2 also permits the use of paging when operating in V86 mode. Using paging, the 1-MByte address space of the V86 task can be mapped to anywhere in the 4-GByte linear address space of the ST486DX/DX2 CPU.

The paging hardware allows multiple V86 tasks to run concurrently, and provides protection and operating system isolation. The paging hardware must be enabled to run multiple V86 tasks or to relocate the address space of a V86 task to physical address space greater than 1 MByte.

2.9.2 Protection

All V86 tasks operate with the least amount of privilege (level 3) and are subject to all of the ST486DX/DX2 protected mode protection checks. As a result, any attempt to execute a privileged instruction within a V86 task results in a general protection fault.

In V86 mode, a slightly different set of instructions are sensitive to the I/O privilege level (IOPL) than in protected mode. These instructions are: CLI, INT n, IRET, POPF, PUSHF, and STI. The INT3, INTO and BOUND variations of the INT instruction are not IOPL sensitive.

2.9.3 Interrupt Handling

To fully support the emulation of an 8086-type machine, interrupts in V86 mode are handled as follows. When an interrupt or exception is serviced in V86 mode, program execution transfers to the interrupt service routine at privilege level 0 (i.e., transition from V86 to protected mode occurs) and the VM bit in the EFLAGS register is cleared. The protected mode interrupt service routine then determines if the interrupt came from a protected mode or V86 application by examining the VM bit in the EFLAGS image stored on the stack. The interrupt service routine may then choose to allow the 8086 operating system to handle the interrupt or may emulate the function of the interrupt handler. Following completion of the interrupt service routine, an IRET instruction restores the EFLAGS register (restores VM=1) and segment selectors and control returns to the interrupted V86 task.

2.9.4 Entering and Leaving V86 Mode

V86 mode is entered from protected mode by either executing an IRET instruction at CPL = 0 or by task switching. If an IRET is used, the stack must contain an EFLAGS image with VM = 1. If a task switch is used, the TSS must contain an EFLAGS image containing a 1 in the VM bit position. The POPF instruction cannot be used to enter V86 mode since the state of the VM bit is not affected. V86 mode can only be exited as the result of an interrupt or exception. The transition out must use a 32-bit trap or interrupt gate which must point to a non-conforming privilege level 0 segment (DPL = 0), or a 32-bit TSS. These restrictions are required to permit the trap handler to IRET back to the V86 program.

2.10 FPU Operations

2.10.1 FPU Register Set

In addition to the registers described to this point, the FPU circuitry within the ST486DX/DX2 provides the user eight data registers accessed in a stack-like manner, a control register, and a status register. The ST486DX/DX2 also provides a data register tag word which improves context switching and stack performance by maintaining empty/non-empty status for each of the eight data registers. In addition, registers in the CPU contain pointers to (a) the memory location containing the current instruction word and (b) the memory location containing the operand associated with the current instruction word (if any).

FPU Tag Word Register. The ST486DX/DX2 maintains a tag word register (Figure 2-34) comprised of two bits for each physical data register. Tag Word fields assume one of four values depending on the contents of their associated data registers, Valid (00), Zero (01), Special (10), and Empty (11). Note: Denormal, Infinity, QNaN, SNaN and unsupported formats are tagged as "Special". Tag values are maintained transparently by the ST486DX/DX2 and are only available to the programmer indirectly through the FSTENV and FSAVE instructions. The tag word with tag fields for each associated physical register, tag(n), is shown in Figure 2-34.

FPU Status Register. The FPU circuitry communicates information about its status and the results of operations to the ST486DX/DX2 via the status register. The FPU status register (Figure 2-35, Page 82) is comprised of bit fields (Table 2-29, Page 82) that reflect exception status, operation execution status, register status, operand class, and comparison results. This register is continuously accessible to the ST486DX/DX2 CPU regardless of the state of the Control or Execution Units.

FPU Mode Control Register. The FPU Mode Control Register (MCR) (Figure 2-36, Page 83) is used by the CPU to specify the operating mode of the FPU. The MCR contains bit fields (Table 2-30, Page 83) which specify the rounding mode to be used, the precision by which to calculate results, and the exception conditions which should be reported to the CPU via traps. The user controls precision, rounding, and exception reporting by setting or clearing appropriate bits in the MCR.

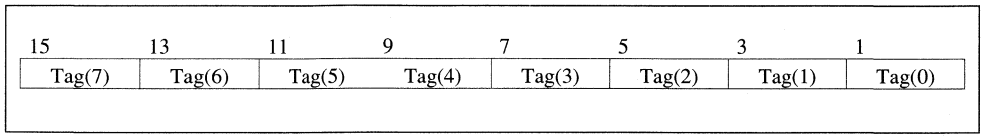


Figure 2 - 34. Tag Word Register

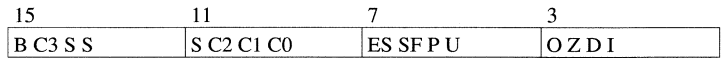


Figure 2 - 35. Status Register

Table 2 - 29. Status Control Register Bit Definitions

BIT POSITION	NAME	DESCRIPTION
15	B	Copy of the ES bit. (ES is bit 7 in this table.)
14, 10 - 8	C3 - C0	Condition code bits.
13 - 11	SSS	Top of stack register number which points to the current TOS.
7	ES	Error indicator. Set to 1 if an unmasked exception is detected.
6	SF	Stack Fault or invalid register operation bit.
5	P	Precision error exception bit.
4	U	Underflow error exception bit.
3	O	Overflow error exception bit.
2	Z	Divide by zero exception bit.
1	D	Denormalized operand error exception bit.
0	I	Invalid operation exception bit.

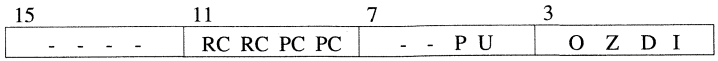


Figure 2 - 36. Mode Control Register

Table 2 - 30. Mode Control Register Bit Definitions

BIT POSITION	NAME	DESCRIPTION
11 - 10	RC	Rounding Control bits: 00 Round to nearest or even 01 Round towards minus infinity 10 Round towards plus infinity 11 Truncate
9 - 8	PC	Precision Control bits: 00 24-bit mantissa 01 Reserved 10 53-bit mantissa 11 64-bit mantissa
5	P	Precision error exception bit mask.
4	U	Underflow error exception bit mask.
3	O	Overflow error exception bit mask.
2	Z	Divide by zero exception bit mask.
1	D	Denormalized operand error exception bit mask.
0	I	Invalid operation exception bit mask.

BUS INTERFACE

3.0 BUS INTERFACE

3.1 Overview

The following sections describe the ST486DX/DX2 input and output signals. The signals are described in the same order as they are listed in Figure 3-1. The discussion of these signals is arranged by functional groups. These groups are indicated in Figure 3-1 and listed in Table 3-1 (Page 88).

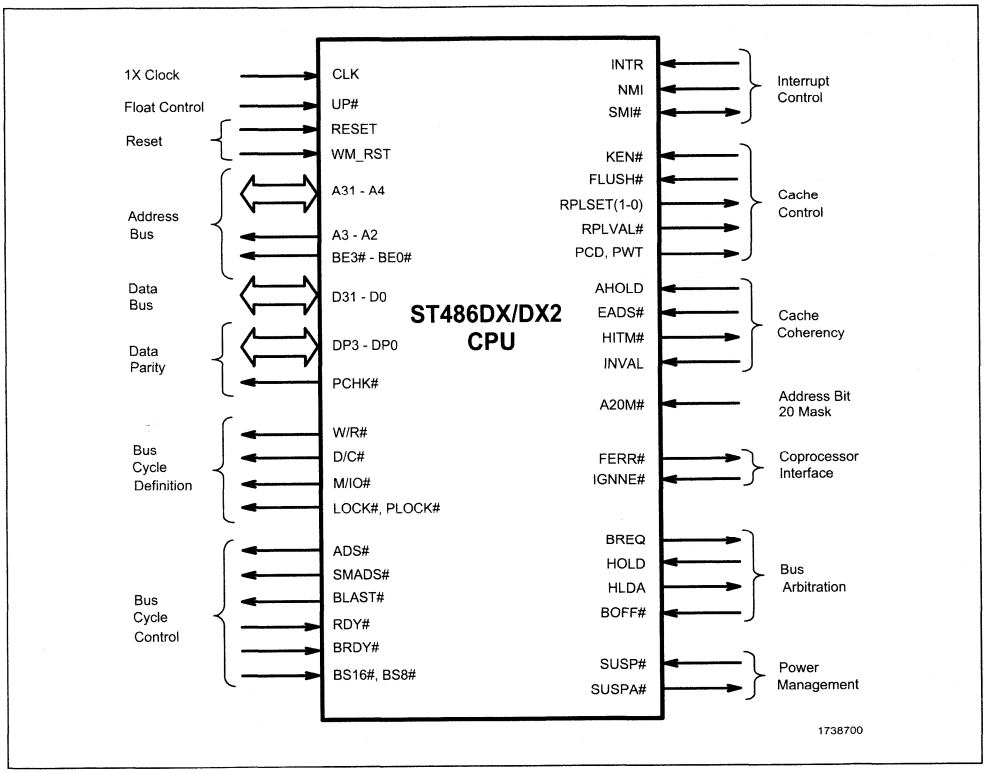


Figure 3 - 1. ST486DX/DX2 Functional Signal Groupings

Table 3 - 1. ST486DX/DX2 Signal Summary

SIGNAL	SIGNAL NAME	SIGNAL GROUP
A20M#	Address Bit 20 Mask	--
A31-A2	Address Bus Lines	Address Bus
ADS#	Address Strobe	Bus Cycle Control
AHOLD	Address Hold	Cache Coherency
BE3#-BE0#	Byte Enables	Address Bus
BLAST#	Burst Last	Bus Cycle Control
BOFF#	Back-Off	Bus Arbitration
BRDY#	Burst Ready	Bus Cycle Control
BREQ	Bus Request	Bus Arbitration
BS8#	Bus Size 8	Bus Cycle Control
BS16#	Bus Size 16	Bus Cycle Control
CLK	1X Clock Input	--
D(31-0)	Data Bus Lines	Data Bus
D/C#	Data/Control	Bus Cycle Definition
DP(3-0)	Data Parity Bits	Data Parity
EADS#	External Address Strobe	Cache Coherency
FERR#	Floating Point Error	Coprocessor Interface
FLUSH#	Cache Flush	Cache Control
HITM#	Hit on Modified Line	Cache Coherency
HLDA	Hold Acknowledge	Bus Arbitration
HOLD	Hold Request	Bus Arbitration
IGNNE#	Ignore Numeric Error	Coprocessor Interface
INTR	Maskable Interrupt Request	Interrupt Control
INVAL	Invalidate	Cache Coherency
KEN#	Cache Enable	Cache Control
LOCK#	Bus Lock	Bus Cycle Definition
M/IO#	Memory/Input-Output	Bus Cycle Definition
NMI	Non-Maskable Interrupt Request	Interrupt Control
PCD	Page Cache Disable	Cache Control
PCHK#	Parity Check Status	Data Parity
PLOCK#	Pseudo Bus Lock	Bus Cycle Definition
PWT	Page Write Through	Cache Control
RDY#	Bus Ready	Bus Cycle Control
RESET	Cold Reset	Reset Control
RPLSET(1-0)	Replacement Set Bits	Cache Control
RPLVAL#	Replacement Set Valid	Cache Control
SMADS#	SMM Address Strobe	Bus Cycle Control
SMI#	System Management Interrupt	Interrupt Control
SUSP#	Suspend Request	Power Management
SUSPA#	Suspend Acknowledge	Power Management
UP#	Upgrade Present	Float Control
WM_RST	Warm Reset	Reset Control
W/R#	Write/Read	Bus Cycle Definition

The pound symbol (#) following a signal name indicates that when the signal is in its active (asserted) state, the signal is at a logic low level. When the "#" is not present at the end of a signal name, the logic high level represents the active state. The following two sections describe the signals and their functional timing characteristics. Additional signal information may be found in Chapter 4, Electrical Specifications. Chapter 4 documents the DC and AC characteristics for the signals including voltage levels, propagation delays, setup times and hold times. Specified setup and hold times must be met for proper operation of the ST486DX/DX2.

3.2 Signal Descriptions

3.2.1 Clock Input

The **Clock Input (CLK)** signal is the basic timing reference for the ST486DX. This signal is also used as an input to time the ST486DX2, but the signal is first doubled within the ST486DX2 to provide an internal 2x CPU clock. For both CPUs, the CLK signal controls external CPU bus timing. The rising edge of the CLK signal defines the starting point for measurement of external AC specifications for both CPUs.

3.2.2 Upgrade Provision

Upgrade Present (UP#) is an active low input that forces the ST486DX/DX2 to float (three-state) all outputs and enter a power-down state.

3.2.3 Reset Control

Reset (RESET) is an active high input signal that, when asserted, suspends all operations in progress and places the ST486DX/DX2 into a reset state. RESET is an asynchronous signal but must meet specified setup and hold times to guarantee recognition at a particular clock edge. While RESET is active only the HOLD input signal is recognized. The ST486DX/DX2 output signals are initialized to their reset state during the internal processor reset sequence. The ST486DX/DX2 reset signal states are shown in Table 3-2 (Page 90).

Warm Reset (WM_RST) is an active high input signal that, when asserted, suspends all operations in progress and places the ST486DX/DX2 in a known state. WM_RST is an asynchronous signal but must meet specified setup and hold times to guarantee recognition at a particular clock edge. WM_RST differs from RESET in that the valid and dirty bits in the on-chip cache, and the bits in the Configuration Registers remain unchanged. If RESET and WM_RST are asserted simultaneously, WM_RST is ignored and RESET takes priority. WM_RST is ignored following RESET and can be enabled using the WBAK bit in CCR2. WM_RST has the same timing and duration specifications as RESET.

Neither RESET nor WM_RST should be asserted during SMM as system hardware might not be returned to a known state if the SMI handler is not allowed to complete.

Table 3 - 2. Signal States During Reset

SIGNAL	SIGNAL STATE DURING RESET/WM_RST	SIGNAL	SIGNAL STATE DURING RESET/WM_RST
A20M#	Ignored	INVAL	Ignored
A31-A2	Undefined	KEN#	Ignored
ADS#	1	LOCK#	1
AHOLD	Sampled for Self-Test Initiation	M/IO#	Undefined
BE3#-BE0#	Undefined	NMI	Ignored
BLAST#	Undefined	PCD	Undefined
BOFF#	Recognized	PCHK#	1
BRDY#	Ignored	PEREQ	Ignored
BREQ	0	PLOCK#	1
BS8#	Ignored	PWT	Undefined
BS16#	Ignored	RDY#	Ignored
BUSY#	Ignored	RESET	Recognized
D(31-0)	Float	RPLSET(1-0)	Float/Undefined
D/C#	Undefined	RPLVAL#	Float/1
DP(3-0)	Float	SMADS#	Float
EADS#	Ignored	SMI#	Float
ERROR#	Ignored	SUSP#	Ignored
FLUSH#	Ignored	SUSPA#	Float
HLDA	Responds to HOLD	UP#	Recognized
HOLD	Recognized	WM_RST	Ignored
INTR	Ignored	W/R#	Undefined

3.2.4 Address Bus

The **Address Bus (A31-A2)** signals provide physical memory or I/O port addresses. A31-A4 are bi-directional signals used by the ST486DX/DX2 to drive addresses to memory and I/O devices, and used by the system logic to drive cache inquiry addresses into the processor. A3 and A2 are output signals only and are ignored during cache inquiry cycles. All address lines can be used for addressing physical memory allowing a 4GByte address space. During I/O port accesses A31-A16 are driven low. This allows for a 64 KByte I/O address space.

If the COP bit in Configuration Control Register 2 (CCR2) is set, the ST486S/S2 issues coprocessor I/O accesses on the address bus. During coprocessor accesses, A30-A16 are driven low and A31 is driven high. Consequently, the ST486DX/DX2 drives I/O address 8000 00F8h for coprocessor command transfers and 8000 00Fch for coprocessor data transfers. A31-A2 float during bus or address hold states.

The **Byte Enables (BE3#-BE0#)** are active low three-state outputs that determine which bytes within the 32-bit data bus are required for the current memory or I/O access as shown in Table 3-2 (Page 90). During the first cycle of a cache line fill, the ST486DX/DX2 expects data to be returned as if all data bytes are enabled regardless of the state of the byte enable outputs. BE3#-BE0# float during bus hold states.

3.2.5 Data Bus

The **Data Bus** (D31-D0) signals are three-state bi-directional signals which provide the data path between the ST486DX/DX2 and external memory and I/O devices. The data bus inputs data during memory read, I/O read and interrupt acknowledge cycles and outputs data during memory and I/O write cycles. Data read operations require that specified data setup and hold times be met for correct operation. The data bus signals are high active and float while the CPU is in a hold acknowledge state.

3.2.6 Data Parity

The **Data Parity Bus (DP3-DP0)** signals are the four three-state bi-directional signals which provide the parity associated with the four-byte data bus. There is one data parity bit per data byte as shown in Table 3-4. Even parity is driven on DP(3-0) for all data write cycles. During read cycles, DP(3-0) are read by the ST486DX/DX2 and are used with the corresponding data bus pins to check for even parity.

Table 3 - 4. Parity Bit-Data Bus Correlation

PARITY BIT	DATA BYTE
DP3	D31 - D24
DP2	D23 - D16
DP1	D15 - D8
DP0	D7 - D0

Parity Check (PCHK#) is an active low output used to indicate that a parity error has occurred on a read operation. Parity is checked for all reads except interrupt acknowledge cycles and coprocessor I/O cycles, and is only checked for valid bytes as indicated by the byte enable outputs and the bus size inputs. PCHK# is only valid during the clock immediately after read data is returned to the ST486DX/DX2 and is inactive otherwise. Parity errors signaled by a logic low on PCHK# have no effect on processor execution.

3.2.7 Bus Cycle Definition

The bus cycle definition signals consist of five three-state outputs (M/IO#, D/C#, W/R#, LOCK#, PLOCK#). These outputs define the bus cycle type for the operation being performed as listed in Table 3-5 (Page 92). M/IO#, D/C#, W/R# and LOCK# are the primary bus cycle definition signals and are valid when address strobe (ADS#) is active. PLOCK# is a function of the cache enable input and, therefore, is valid only in the final clock of the bus cycle. The bus cycle definition signals are active low and float during bus hold states.

Memory/IO (M/IO#) distinguishes between memory and I/O operations. When high, this signal indicates that the current bus cycle is a memory read or memory write. When low, M/IO# indicates that the current bus cycle is an I/O read, I/O write, interrupt acknowledge cycle or special bus cycle.

Data/Control (D/C#) distinguishes between data and control operations. When high, this signal indicates that the current bus cycle is a data transfer to or from memory or I/O. When low, D/C# indicates that the current bus cycle involves a control function such as a halt, interrupt acknowledge or code fetch.

Write/Read (W/R#) distinguishes between write and read operations. When high, this signal indicates that the current bus cycle is a memory write, I/O write or a special bus cycle. When low, this signal indicates that the current cycle is a memory read, I/O read or interrupt acknowledge cycle.

Table 3 - 5. Bus Cycle Types

M/IO#	D/C#	W/R#	LOCK#	BUS CYCLE TYPE
0	0	0	0	Interrupt Acknowledge
0	0	0	1	--
0	0	1	0	--
0	0	1	1	Special Cycle: BE3# - BE0# = 1110: Shutdown BE3# - BE0# = 1101: Flush BE3# - BE0# = 1011: Halt BE3# - BE0# = 0111: Write-Back
0	1	X	0	--
0	1	0	1	I/O Data Read
0	1	1	1	I/O Data Write
1	0	X	0	--
1	0	0	1	Memory Code Read
1	0	1	1	--
1	1	0	0	Locked Memory Data Read
1	1	0	1	Memory Data Read
1	1	1	0	Locked Memory Data Write
1	1	1	1	Memory Data Write
X = don't care, -- = does not occur				

Bus Lock (LOCK#) is an active low output which, when asserted, indicates that other system bus masters are denied access to control of the CPU bus. The LOCK# signal may be explicitly activated during bus operations by including the LOCK prefix on certain instructions. LOCK# is always asserted during descriptor updates, interrupt acknowledge sequences and when executing the XCHG instruction. The ST486DX/DX2 does not enter the hold acknowledge state in response to HOLD while the LOCK# output is active.

Pseudo-Lock (PLOCK#) is an active low output that is asserted during reads and writes to/from memory that are greater than 32 bits and therefore require multiple bus cycles to complete. The ST486DX/DX2 asserts PLOCK# during segment descriptor reads (64 bits), cache line fills (128 bits) and non-cacheable prefetches (128 bits). The ST486DX/DX2 does not enter the hold acknowledge state in response to HOLD while the PLOCK# input is active, except during non-cacheable, non-burstable code prefetches. Under this condition, the ST486DX/DX2 acknowledges HOLD on bus cycle boundaries even though PLOCK# is asserted.

3.2.8 Bus Cycle Control

The bus cycle control signals (ADS#, BLAST#, RDY#, BRDY#, BS16#, BS8#, SMADS#) allow the ST486DX/DX2 to indicate the beginning of a bus cycle and allows system hardware to control bus cycle termination timing, bursting and bus sizing.

Address Strobe (ADS#) is an active low output which indicates that the ST486DX/DX2

has driven a valid address and bus cycle definition on the appropriate output pins. If the current cycle is a memory access, ADS# also indicates that the current bus cycle is intended for normal memory space rather than system management memory. ADS# floats during bus hold states.

SMM Address Strobe (SMADS#) is asserted instead of the ADS# during SMM bus cycles and indicates that SMM memory is being accessed. SMADS# floats while the CPU is in a hold acknowledge or float state. The SMADS# output is disabled (floated) following reset and can be enabled using the SMI bit in the CCR1 configuration register.

Burst Last (BLAST#) is an activc low output which indicates that the current 32-bit data transfer is either the last transfer of a multiple transfer cycle or a single transfer cycle. BLAST# is valid for the second and subsequent clock cycles within both burstable and non-burstable cycles. BLAST# floats during bus hold states.

Non-Burst Ready (RDY#) is an active low input which is driven by the system to indicate that the current bus cycle can be terminated. During a read cycle, assertion of RDY# indicates that the system hardware has presented valid data to the CPU. When RDY# is sampled active, the ST486DX/DX2 latches the input data and terminates the cycle. During a write cycle, RDY# assertion indicates that the system hardware has accepted the ST486DX/DX2 output data. RDY# is active during address hold states.

Burst Ready (BRDY#) is an active low input which is driven by the system to indicate that the current transfer within a burst cycle can be terminated. The ST486DX/DX2 samples BRDY# in the second and subsequent clocks of a multiple transfer cycle. If BRDY# is returned instead of RDY# for the first transfer of a multiple transfer cycle, the ST486DX/DX2 completes the remaining transfers as a burst cycle. BRDY# must be returned instead of RDY# for each transfer (except the final transfer) to maintain the burst cycle. If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is aborted. BRDY# is active during address hold states.

The ST486DX/DX2 is capable of bursting code fetches, memory data reads, memory data writes with BS16# or BS8# active, or cache line write-back cycles if the BWRT bit is set in CCR2. The ST486DX/DX2 only bursts cache line write-back cycles if all four doublewords within the cache line have been modified and need to be written back to memory or if the write-back cycles are the results of a cache inquiry cycle. If three or less doublewords have been modified, the ST486DX/DX2 issues the write-back cycles as non-bursted 32-bit write cycles.

Bus Size 16 (BS16#) and **Bus Size 8 (BS8#)** are active low inputs that allow connection of the 32-bit CPU data bus to an external bus of either 16 or 8 bits. When these inputs are asserted, the ST486DX/DX2 performs multiple bus cycles to complete a single 32-bit transfer. BS16# and BS8# are sampled each clock and the state of these pins in the clock before ready is returned is used to determine the bus size for the current cycle. Table 3-6 (Page 94) lists the data pins read during transfers with BS16# or BS8# active. If both BS8# and BS16# are asserted, BS8# is used. During write cycles, valid data is only driven on the data pins corresponding to the active byte enables.

Table 3 - 6. Data Pins Read during BS16# and BS8# Transfers

BE3#	BE2#	BE1#	BE0#	WITH BS16#	WITH BS8#
1	1	1	0	D7 - D0	D7 - D0
1	1	0	1	D15 - D8	D15 - D8
1	0	1	1	D23 - D16	D23 - D16
0	1	1	1	D31 - D24	D31 - D24
1	1	0	0	D15 - D0	D7 - D0
1	0	0	1	D15 - D8	D15 - D8
0	0	1	1	D31 - D16	D23 - D16
1	0	0	0	D15 - D0	D7 - D0
0	0	0	1	D15 - D8	D15 - D8
0	0	0	0	D15 - D0	D7 - D0

3.2.9 Interrupt Control

The interrupt control input signals (INTR, NMI, SMI#) allow the execution of the ST486DX/DX2 current instruction stream to be interrupted and suspended.

Maskable Interrupt Request (INTR) is a level-sensitive input which causes the processor to suspend execution of the current instruction stream and begin execution of an interrupt service routine. The INTR input can be masked (ignored) through the Flags Register IF bit. When unmasked, the ST486DX/DX2 responds to the INTR input by issuing two locked interrupt acknowledge cycles. During the second interrupt acknowledge cycle, the ST486DX/DX2 reads an 8-bit value, the interrupt vector, from an external interrupt controller. The 8-bit interrupt vector indicates the interrupt level that caused generation of the INTR and is used by the CPU to determine the beginning address of the interrupt service routine. To assure recognition of the INTR request, INTR must remain active until the start of the first interrupt acknowledge cycle.

Non-maskable Interrupt Request (NMI) is a rising-edge sensitive input which causes the processor to suspend execution of the current instruction stream and begin execution of an NMI interrupt service routine. The NMI interrupt request cannot be masked by software. Asserting NMI causes an interrupt which internally supplies interrupt vector 2h to the CPU core. Therefore, external interrupt acknowledge cycles are not necessary.

System Management Interrupt (SMI#) is a bidirectional signal and level-sensitive interrupt with higher priority than the NMI interrupt. SMI# must be active for at least one clock period to be recognized by the ST486DX/DX2. After the SMI# interrupt is acknowledged, the SMI# pin is driven low by the ST486DX/DX2 for the duration of the SMI service routine. The SMI# input is ignored following reset and can be enabled using the SMI bit in the CCR1 configuration register.

3.2.10 Cache Control

The cache control signals (KEN#, FLUSH#, RPLSET(1-0), RPLVAL#, PCD, PWT) are used to indicate cache status and control caching activity.

Cache Enable (KEN#) is an active low input which indicates that the data being returned during the current cycle is cacheable. When the ST486DX/DX2 is performing a cacheable code fetch or memory data read cycle and KEN# is sampled asserted one clock before the first BRDY# or RDY#, the cycle is transformed into a 16-byte cache line fill. Returning KEN# active one clock before ready is returned during the last read in the cache line fill causes the line to be written to the on-chip cache. I/O accesses, locked reads, system management memory accesses and interrupt acknowledge cycles are never cached.

Cache Flush (FLUSH#) is an active low input that forces the ST486DX/DX2 to invalidate the entire cache contents while in write-through cache mode. If the cache is operating in write-back mode, FLUSH# forces the ST486DX/DX2 to write-back all dirty data in the cache. If the INVAL pin is asserted when FLUSH# is sampled, the ST486DX/DX2 also invalidates the cache contents following the write-back of dirty data. FLUSH# need only be asserted for a single clock but must meet specified setup and hold times to guarantee recognition at a particular clock edge.

The **Replacement Set (RPLSET1, RPLSET0)** outputs indicate which set in the cache is currently undergoing a line replacement. RPLSET1 and RPLSET0 are disabled (three-state) following RESET and is enabled by setting the RPL bit in the CCR1 configuration register.

Replacement Set Valid (RPLVAL#) is an active low output driven during a cache line fill cycle to indicate that RPLSET(1-0) are valid for the current cycle. RPLVAL# and RPLSET(1-0) provide external hardware the capability of monitoring the cache LRU replacement algorithm. RPLVAL# is disabled (three-state) following RESET and is enabled by setting the RPL bit in the CCR1 configuration register.

Page Cache Disable (PCD) is an active high output that reflects the state of the PCD page attribute bit in the page table entry or the page directory entry. If paging is disabled or for cycles that are not paged, the PCD pin is driven low. PCD is masked by the cache disable (CD) bit in CR0 and floats during bus hold states.

Page Write Through (PWT) is an active high output that reflects the state of the PWT page attribute bit in the page table entry or the page directory entry. If paging is disabled or for cycles that are not paged, the PWT pin is driven low. PWT floats during bus hold states.

3.2.11 Cache Coherency

The cache coherency signals (AHOLD, EADS#, HITM#, INVAL) are used to initiate and monitor cache inquiry cycles. HITM# and INVAL are only required when operating the ST486DX/DX2 cache in write-back mode.

Address Hold (AHOLD) Request is an active high input which forces the ST486DX/DX2 to float A31-A2 in the next clock cycle. While AHOLD is asserted, only the address bus is disabled. The current bus cycle remains active and can be completed in the normal fashion. The ST486DX/DX2 does not generate additional bus cycles while AHOLD is asserted, except cache line write-back cycles in response to a cache inquiry.

The ST486DX/DX2 also samples AHOLD during RESET. If AHOLD is asserted at the clock edge prior to the falling edge of RESET, the ST486DX/DX2 executes its built-in self-test prior to issuing any bus cycles.

External Address Strobe (EADS#) is an active low input used to indicate to the ST486DX/DX2 that a valid cache inquiry address is being driven on the ST486DX/DX2 address bus (A31-A4). The ST486DX/DX2 checks the on-chip cache for this address. If the cache is operating in write-through mode and is guaranteed to have no dirty locations, the cache line corresponding to the specified address is invalidated if present in the cache. If the cache is operating in write-back mode or contains dirty data from previous write-back operation, the cache line corresponding to the specified address is first checked for dirty data. If dirty data exists, the dirty data is first written to external memory. The state of the INVAL pin at the time EADS# is sampled active determines the final state of the cache line.

A cache inquiry cycle using EADS# may be run with the ST486DX/DX2 in either an address hold or bus hold state and the inquiry address driven by an external device. Additionally, an inquiry cycle may be run while the ST486DX/DX2 is driving the address bus. In this case, the current address is used as the inquiry address.

Hit on Modified Data (HITM#) is an active low output used to indicate that the current cache inquiry address has been found in the cache and dirty data exists in the cache line. HITM# is asserted one clock after EADS# is sampled active, and remains asserted until all dirty data has been written to external memory. The ST486DX/DX2 does not accept additional cache inquiry cycles while HITM# is asserted. HITM# is disabled (floats) following RESET and is enabled by setting the WBAK bit in CCR2.

Invalidate Request (INVAL) is an active high input driven by the system during a cache inquiry cycle to indicate the final state of the cache line if an inquiry hit occurs. INVAL is sampled with EADS# and is only required when operating the ST486DX/DX2 cache in write-back mode. A logic one on INVAL indicates that the final state of the cache line is invalid. A logic zero on INVAL indicates that the final state of the cache line is valid and clean. INVAL is also sampled with the FLUSH# input. In this case, the state of INVAL determines the final state of the entire cache. INVAL is ignored following RESET and is enabled by setting the WBAK bit in CCR2.

3.2.12 Address Bit 20 Mask

Address Bit 20 Mask (A20M#) is an active low input which causes the ST486DX/DX2 to mask (force low) physical address bit 20 when driving the external address bus or when performing an internal cache access. Asserting A20M# emulates the 1 MByte address wrap-around that occurs on the 8086. A20 masking should not be done by external logic. The A20M# input is ignored during three conditions:

- while paging is enabled.
- while writing back dirty cache data to system memory. (This occurs only if the data was loaded into the cache when A20M# was inactive).
- during system management address space accesses.

3.2.13 FPU Interface Signals

The FPU interface signals (FERR#, IGNNE#) are used to control error reporting for the on-chip floating point unit. These signals are typically used for a PC-compatible system implementation. For other applications, FPU errors are reported to the ST486DX/DX2 CPU core through an internal interface.

Floating Point Error Status (FERR#) is an active low output asserted by the ST486DX/DX2 when an unmasked floating point error occurs. FERR# is asserted during execution of the FPU instruction that caused the error. FERR# does not float during bus hold states.

Ignore Numeric Error (IGNNE#) is an active low input which forces the ST486DX/DX2 to ignore any pending unmasked FPU errors and allows continued execution of floating point instructions. When IGNNE# is not asserted and an unmasked FPU error is pending, the ST486DX/DX2 will only execute the following floating point instructions: FNCLEX, FNINIT, FNSAVE, FNSTCW, FNSTENV, and FNSTSW. IGNNE# is ignored when the NE bit in CR0 is set to a 1. This and related actions are detailed below using pseudo-code.

```
if FERR#=0 then
        {
        if NE =1 then
            generate interrupt 16 at next FPU
            or WAIT instruction
        else
            {
            if IGNNE#=0 then
                continue execution
            else
                while IGNNE# AND (FPU
                    instruction OR  WAIT instruction),
                    wait until IGNNE# goes active.
                    CPU stalls on a FPU instruction
                    or WAIT instruction.  CPU
                    continues to service interrupts.
            }
        }
```

3.2.14 Bus Arbitration

The bus arbitration (BREQ, HOLD, HLDA, BOFF#) signals allow the ST486DX/DX2 to relinquish control of its local bus when requested by another bus master device. Once the processor has relinquished its bus (three-stated), the bus master device can then drive the local bus signals.

Bus Request (BREQ) is an active high output asserted by theST486DX/DX2 whenever a bus cycle is pending internally. The ST486DX/DX2 always asserts BREQ in the first clock of a bus cycle as well as during bus hold and address hold states if a bus cycle is pending. If no additional bus cycles are pending, BREQ is negated prior to termination of the current cycle.

Bus Hold Request (HOLD) is an active high input used to indicate that another bus master requests control of the CPU's local bus. After recognizing the HOLD request and completing the current bus cycle, burst cycle, cache line fill, sequence of locked bus cycles, or cache line write-back, the ST486DX/DX2 responds by floating the local bus and asserting the hold acknowledge (HLDA) output. The bus remains granted to the requesting bus master until HOLD is negated. Once HOLD is sampled negated, the ST486DX/DX2 simultaneously drives the local bus and negates HLDA.

Hold Acknowledge (HLDA) is an active high output used to indicate that the ST486DX/DX2 has responded to the HOLD input and has relinquished control of its local bus. Table 3-7 (Page 99) lists the state of all the ST486DX/DX2 signals during a bus hold state. The ST486DX/DX2 continues to operate during bus hold states as long as the on-chip cache can satisfy bus requests. HLDA is asserted until HOLD is negated. Once HOLD is sampled negated, the ST486DX/DX2 simultaneously drives the local bus and negates HLDA.

Back-Off (BOFF#) is an active low input that forces the ST486DX/DX2 to abort the current bus cycle and relinquish control of the CPU's local bus in the next clock. The ST486DX/DX2 responds to BOFF# by entering the bus hold state as listed in Table 3-7 (Page 99), but the HLDA output is not asserted. The ST486DX/DX2 remains in bus hold until BOFF# is negated. Once BOFF# is negated, the ST486DX/DX2 restarts the aborted bus cycle. The ST486DX/DX2 ignores any data returned while BOFF# is asserted.

Table 3 - 7. Signal States During Bus Hold

SIGNAL	SIGNAL STATE DURING BUS HOLD	SIGNAL	SIGNAL STATE DURING BUS HOLD
A20M#	Recognized Internally	INVAL	Recognized
A31-A2	Float	IGNNE#	Ignored
ADS#	Float	KEN#	Ignored
AHOLD	Ignored	LOCK#	Float
BE3#-BE0#	Float	M/IO#	Float
BLAST#	Float	NMI	Recognized
BOFF#	Recognized	PCD	Float
BRDY#	Ignored	PCHK#	Driven
BREQ	Driven	PLOCK#	Float
BS8#	Ignored	PWT	Float
BS16#	Ignored	RDY#	Ignored
D(31-0)	Float	RESET	Recognized
D/C#	Float	RPLSET(1-0)	Driven
DP(3-0)	Float	RPLVAL#	Driven
EADS#	Recognized	SMADS#	Float
FERR#	Driven	SMI#	Recognized
FLUSH#	Recognized	SUSP#	Recognized
HITM#	Driven	SUSPA#	Driven
HLDA	Driven	UP#	Recognized
HOLD	Recognized	WM_RST	Recognized
INTR	Recognized	W/R#	Float
Note: HITM#, RPLSET(1-0), RPLVAL#, and SUSPA# are driven during bus hold, only if the appropriate CCR bits are set.			

3.2.15 Power Management Interface

Two power management signals allow the ST486DX/DX2 to enter and exit suspend mode. Suspend mode can also be entered as the result of executing a HLT instruction if the HALT bit in CCR2 is set. Suspend mode circuitry allows the ST486DX/DX2 to consume minimal power while maintaining the entire internal CPU state.

Suspend Request (SUSP#) is an active low input which requests that the ST486DX/DX2 enter suspend mode. After recognition of an active SUSP# input, the processor completes execution of the current instruction, any pending decoded instructions and associated bus cycles. During suspend mode, internal clocks are stopped. With SUSPA# asserted, the external CLK input to the ST486DX/DX2 can be stopped in either phase. Stopping the CLK input further reduces current consumption of the ST486DX/DX2.

To resume operation, the CLK input is restarted (if stopped), followed by negation of the SUSP# input. The processor then resumes instruction fetching and begins execution in the instruction stream at the point it had stopped. The SUSP# input is level sensitive and must meet specified setup and hold times to be recognized at a particular clock edge. The SUSP# input is ignored following reset and can be enabled by setting the SUSP bit in the CCR2 configuration register.

The **Suspend Acknowledge (SUSPA#)** output indicates that the ST486DX/DX2 has entered low-power suspend mode as a result of SUSP# assertion or execution of a HLT instruction. SUSPA# remains asserted until SUSP# is negated, or until an interrupt is serviced if suspend mode was entered via the HLT instruction. The CLK input to the processor may be stopped after SUSPA# has been asserted to further reduce the current consumption of the ST486DX/DX2. The SUSPA# output is disabled (floated) following reset and can be enabled by setting the SUSP bit in the CCR2 configuration register. Table 3-8 (Page 101) shows the state of the ST486DX/DX2 signals when the device is in suspend mode. Inputs are only recognized if the external CLK input is switching.

Table 3 - 8. Signal States During Suspend Mode

SIGNAL NAME	SIGNAL STATE DURING SUSP#/HALT INITIATED SUSPEND MODE
A20M#	Ignored
A31-A2	Driven with previous value.
ADS#	1
AHOLD	Ignored
BE3#-BE0#	Driven with previous value.
BLAST#	Undefined
BOFF#	Recognized
BRDY#	Ignored
BREQ	0
BS8#	Ignored
BS16#	Ignored
D(31-0)	Float
D/C#	Driven with previous value.
DP(3-0)	Float
EADS#	Ignored
FERR#	Ignored
FLUSH#	Ignored
HITM#	1
HLDA	Driven
HOLD	Recognized
IGNNE#	Ignored
INTR	Latched/Recognized
INVAL	Ignored
KEN#	Ignored
LOCK#	1
M/IO#	Driven with previous value.
NMI	Latched/Recognized
PCD	Driven with previous value.
PCHK#	1
PLOCK#	Undefined
PWT	Driven with previous value.
RDY#	Ignored
RESET	Recognized
RPLSET(1-0)	Driven with previous value.
RPLVAL#	1
SMADS#	1
SMI#	Latched/Recognized
SUSP#	Recognized
SUSPA#	0
UP#	Recognized
WM_RST	Recognized
W/R#	0

3.3 Functional Timing

The following functional timing diagrams document operation of pins and functions that do not exist on the 486SX, 487SX, 486DX, or 486DX2 and are unique to the ST486DX/DX2.

3.3.1 Write-Back Cache Coherency

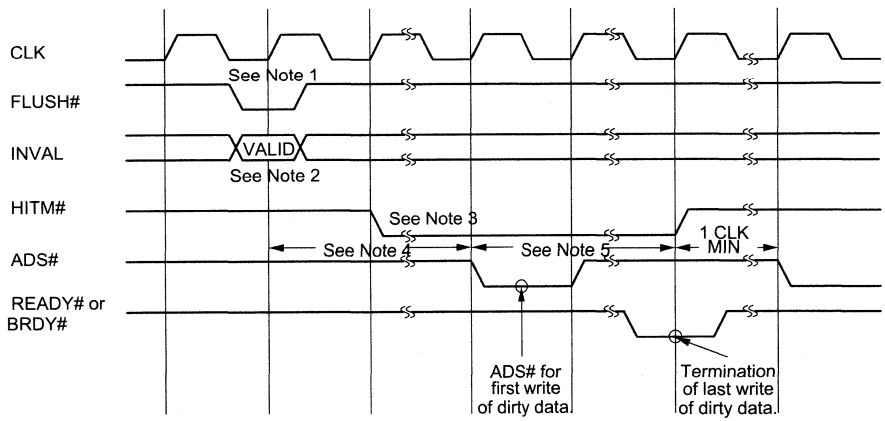
The timing diagrams in this section pertain specifically to the ST486DX/DX2 operating in write-back cache mode with the WBAK bit in CCR2 set to one. If the ST486DX/DX2 cache is currently in write-through mode but has been previously operated in write-back mode, the possibility exists that the most recent or "dirty" data still resides in the cache. Therefore, the following timing diagrams also apply under these circumstances.

Cache coherency must be maintained when previously cached data is modified in or read from external memory by a bus master other than the CPU. A write-back cache differs from a write-through cache in that the most recent data may actually exist in the cache rather than in external memory. If a bus master tries to access data in external memory, the ST486DX/DX2 cache must be checked prior to allowing the bus master to complete the access. If the ST486DX/DX2 contains the most recent data, this data must be written to memory prior to concluding the bus master access.

Several hardware methods exist on the ST486DX/DX2 for checking the cache dirty status and writing the most recent data to external memory. The first two methods, using either the FLUSH# input or the BARB function, cause the ST486DX/DX2 to check each line in the cache and write all dirty data back to memory prior to continuing execution. These two methods are illustrated in Figure 3-2 (Page 103) and Figure 3-3 (Page 104). The third method, using cache inquiry cycles, causes the ST486DX/DX2 to check the cache line specified by the system and then write dirty data back if necessary only for that line. Cache inquiry cycles are illustrated in Figure 3-4 (Page 106), Figure 3-5 (Page 107), Figure 3-6 (Page 108), and Figure 3-7 (Page 109).

Flushing the Cache

The FLUSH# input pin forces the ST486DX/DX2 to write-back all dirty data in the cache. If the INVAL pin is asserted when FLUSH# is sampled, the ST486DX/DX2 also invalidates the cache contents following the write-back of dirty data. The ST486DX/DX2 asserts HITM# in response to FLUSH# if the cache contains dirty data, and keeps HITM# asserted until completion of all required write-back cycles. The ST486DX/DX2 ignores cache inquiry cycles while HITM# is asserted. During write-back of dirty data resulting from a FLUSH# or INVD/WBINVD instructions, HOLD is only acknowledged when (1) the write-back of all dirty data in the current cache line is complete, and (2) the next line in the cache contains no dirty data.

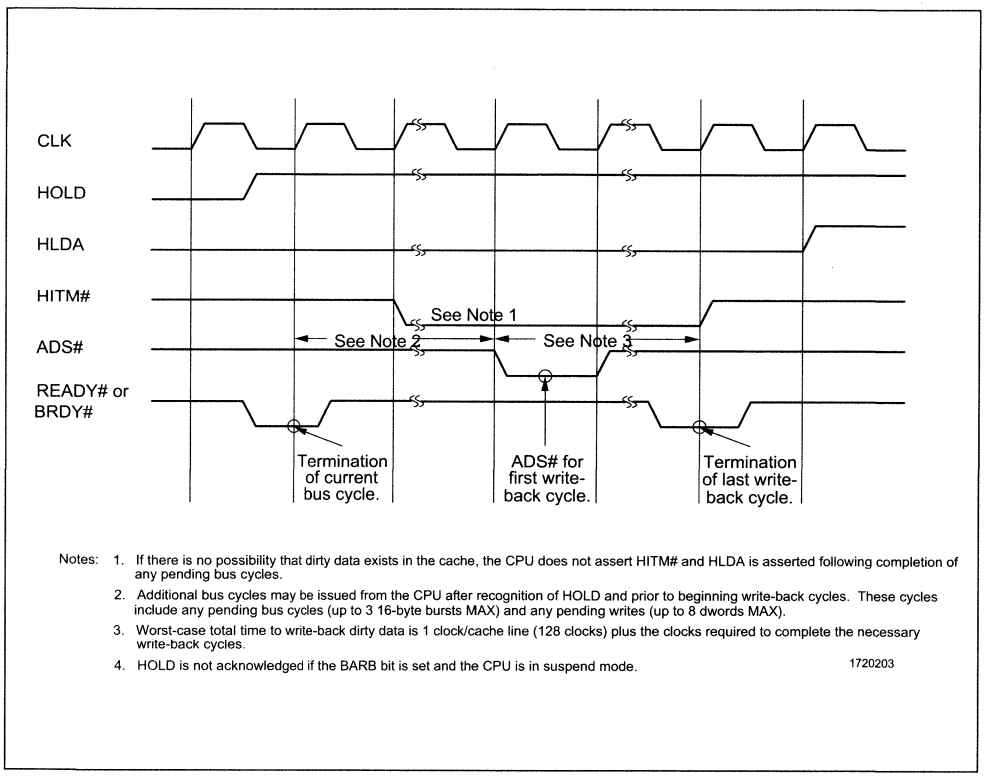


Notes:
1. Asserting FLUSH# stalls the CPU core while in write-back mode. Therefore, FLUSH# should be strobed rather than held low for optimum performance. FLUSH# is ignored while the CPU is in suspend mode (SUSPA# asserted).
2. Final state of cache following write-back of all dirty data is determined by the state of INVAL when FLUSH# is sampled.
3. If there is no possibility that dirty data exists, the CPU does not assert HITM# and bus cycles continue in a normal fashion.
4. Additional bus cycles may be issued from the CPU after recognition of FLUSH# and prior to beginning write-back cycles. These cycles include completion of locked cycles, any pending bus cycles (up to 3 16-byte bursts MAX) and any pending writes (up to 8 dwords MAX).
5. Worst-case total time to write-back dirty data is 1 clock/cache line (128 clocks) plus the clocks required to complete the necessary write-back cycles.

1720103

Figure 3 - 2. FLUSH# Operation During Write-Back Mode

If the BARB bit is set in Configuration Control Register 2 (CCR2), asserting the HOLD input pin forces the ST486DX/2 to write-back all dirty data in the cache prior to assertion of HLDA. The ST486DX/DX2 asserts HITM# in response to HOLD if the cache contains dirty data, and keeps HITM# asserted until completion of all required write-back cycles. The ST486DX/DX2 ignores cache inquiry cycles while HITM# is asserted.



Notes:
1. If there is no possibility that dirty data exists in the cache, the CPU does not assert HITM# and HLDA is asserted following completion of any pending bus cycles.
2. Additional bus cycles may be issued from the CPU after recognition of HOLD and prior to beginning write-back cycles. These cycles include any pending bus cycles (up to 3 16-byte bursts MAX) and any pending writes (up to 8 dwords MAX).
3. Worst-case total time to write-back dirty data is 1 clock/cache line (128 clocks) plus the clocks required to complete the necessary write-back cycles.
4. HOLD is not acknowledged if the BARB bit is set and the CPU is in suspend mode.

1720203

Figure 3 - 3. Write-Back Timing in Response to HOLD with BARB Set

Cache Inquiry Cycles

Cache inquiry cycles are issued by the system with the ST486DX/DX2 in either a bus hold, an address hold, or while the ST486DX/DX2 is driving the address bus. Bus hold is requested by asserting HOLD or BOFF#, and address hold is requested by asserting AHOLD. The system hardware must assert the EADS# input along with a valid state on the INVAL input. If the ST486DX/DX2 is in a bus hold or address hold state, the system must also assert a valid address on A31-A4.

In response, the ST486DX/DX2 checks to see if the specified address is present in the internal cache. If the address is present in the cache, the ST486DX/DX2 checks to see if the corresponding cache line has been modified internally. The ST486DX/DX2 maintains modified or "dirty" status for each 32-bit doubleword (dword) in the 16-byte cache line. If any of the four dwords in the cache line has been marked as dirty, the ST486DX/DX2 asserts the HITM# output and keeps HITM# asserted until the cache line has been written back to external memory.

The ST486DX/DX2 does not issue the write-back cycles until the bus hold state has been exited. However, write-back cycles are issued while the CPU is in an address hold state. The four write-back transfers are issued in sequential address order starting at the +0h address and ending with the +Ch address. Additional cache inquiry cycles are not accepted while HITM# is asserted.

If the cache inquiry cycle occurs with AHOLD asserted and a burst cycle in progress, the four write-back cycles are issued after completion of the burst cycle, or after interruption of the burst cycle by assertion of RDY# instead of BRDY#. If the CPU is currently executing a non-burstable multi-cycle transfer, the write-back cycles are issued after RDY# is received for the current cycle within the transfer. A multi-cycle transfer is defined as a cache fill, non-cacheable prefetch or a transfer requiring multiple cycles due to bus sizing. An interrupted burst read cycle or interrupted non-burstable multi-cycle transfer is restarted in its entirety following completion of the write-back cycles and negation of AHOLD. Interrupted burst write and bus-sized I/O cycles are continued rather than restarted. If the cache inquiry cycle misses in the cache or hits in the cache on a clean line, all interrupted burst cycles and multi-cycle transfers are continued rather than restarted.

If the cache inquiry cycle occurs with BOFF# asserted, the four write-back cycles are issued immediately. When BOFF# is negated, any interrupted transfer, single cycle or multi-cycle, is restarted in its entirety except bus-sized I/O cycles and burst write cycles. Bus-sized I/O cycles are restarted at the cycle that BOFF# interrupted. Burst write cycles are restarted at the 32-bit transfer that BOFF# interrupted.

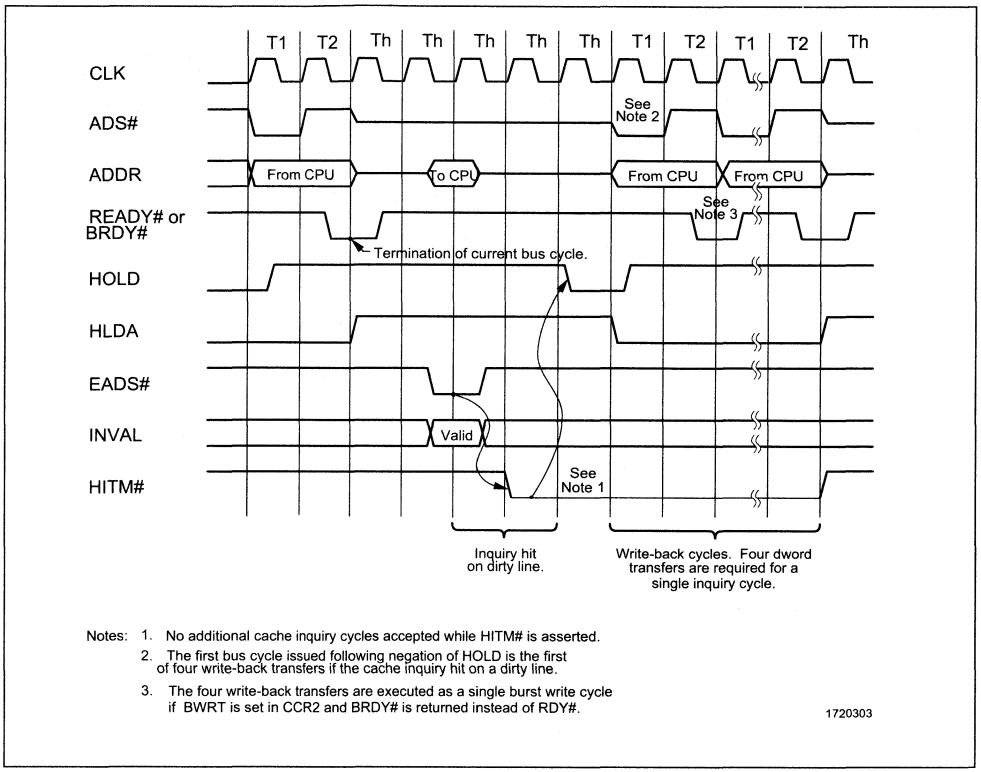
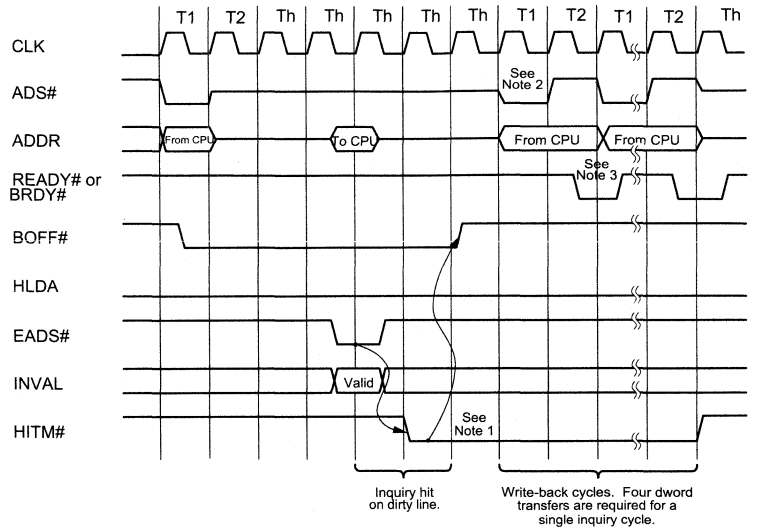


Figure 3 - 4. Inquiry Cycles During Write-Back Mode Using HOLD



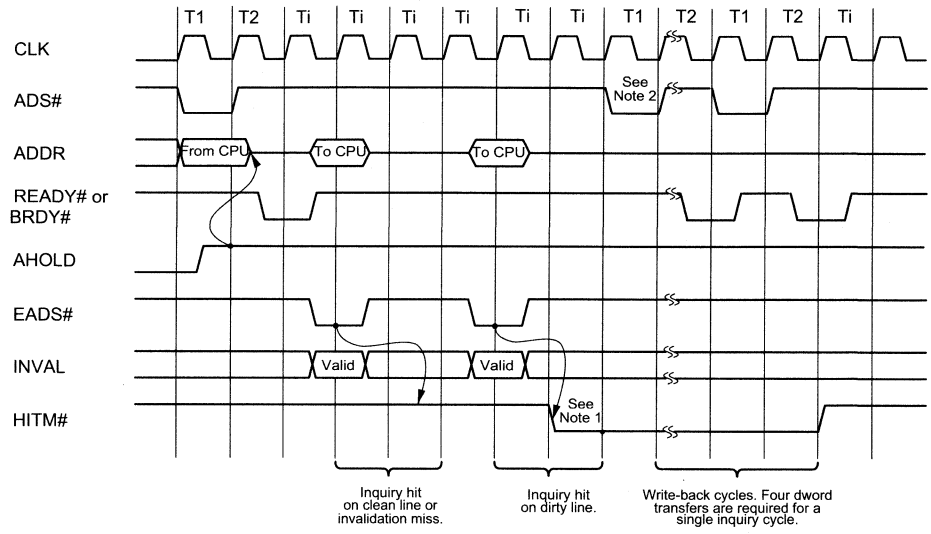
Notes: 1. No additional cache inquiry cycles accepted while HITM# is asserted.

2. The first bus cycle issued following negation of BOFF# is the first of four write-back transfers if the cache inquiry hit on a dirty line. BOFF# must remain asserted until the clock in which HITM# is asserted to guarantee correct cycle ordering.

3. The four write-back transfers are executed as a single burst write cycle if BWRT is set in CCR2 and BRDY# is returned instead of RDY#.

1720900

Figure 3 - 5. Inquiry Cycles During Write-Back Mode Using BOFF#



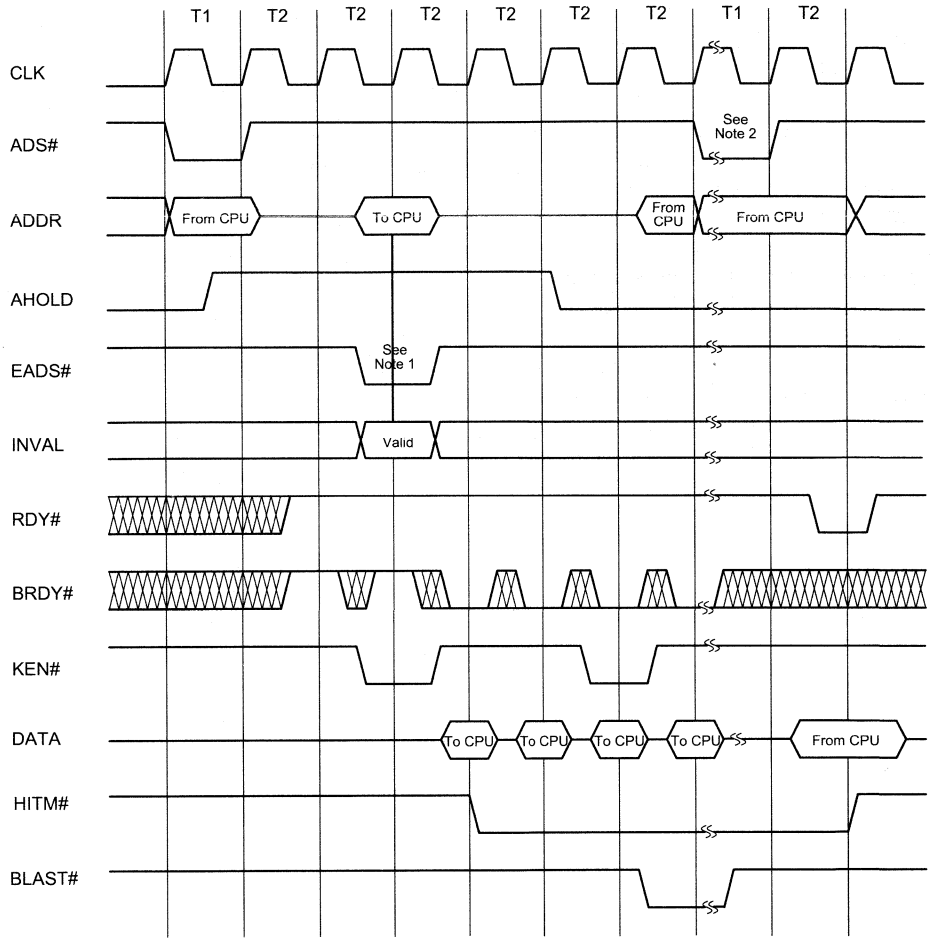
Notes: 1. No additional cache inquiry cycles accepted while HITM# is asserted.

2. The first bus cycle issued following the cache inquiry that hits on a dirty line is the first of four write-back transfers. AHOLD must remain asserted until the clock in which HITM# is asserted to guarantee correct cycle ordering.

3. The four write-back transfers are executed as a single burst write cycle if BWRT is set in CCR2 and BRDY# is returned instead of RDY#.

1721000

Figure 3 - 6. Inquiry Cycles During Write-Back Mode Using AHOLD



Notes: 1. Data returned will not be cached if its address equals the inquiry address for any cache inquiry cycle with EADS# asserted at any clock within the line fill.

2The first bus cycle issued following completion of the bursted line fill is the first of four write-back transfers. AHOLD must remain asserted until the clock in which HITM# is asseted to guarantee correct cycle ordering.

1720503

Figure 3 - 7. Inquiry Cycles During Write-Back Mode Concurrent with Bursted Line Fill

Monitoring Cache Replacement Algorithm

The ST486DX/DX2 RPLVAL# and RPLSET(1-0) signals provide a means for monitoring the LRU replacement algorithm for the on-chip 4-way set associative cache. RPLSET(1-0) indicate which of the 4 sets in the cache is currently undergoing a line replacement. RPLVAL# indicates when the RPLSET(1-0) pins are valid. RPLVAL# asserts on the clock following the first RDY# or BRDY# for any cache fill cycle and remains asserted for one clock. RPLSET(1-0) are driven valid when RPLVAL# is asserted and remain valid for the remainder of the line fill. The data read during a line fill is not placed in the cache (1) if KEN# is negated immediately prior to the last RDY# or BRDY# for the line fill, or (2) if a cache inquiry cycle occurs during the line fill to the line fill address. For these two cases, RPLSET(1-0) are no longer meaningful and therefore, should be ignored.

3.3.2 Burst Write Cycles

The ST486DX/DX2 is capable of burst memory write cycles when the cache is operating in write-back mode. If all four dwords are dirty, burst write cycles are used during line replacement of a dirty cache line and during -write-back cycles in response to a cache flush. Burst write cycles are also used during -write-back cycles in response to a cache inquiry. In this case, all four dwords are always written to memory. The burst write feature is enabled by setting the BWRT bit in Configuration Control Register 2 (CCR2). The control signals used for the burst write cycles are the same as those used for burst read cycles.

Figure 3-8 (Page 111) illustrates a burst write cycle. At the end of the first transfer in the cycle, the external system informs the ST486DX/DX2 that it can perform the burst write by driving BRDY# active. ADS# is driven only during the first address. The last transfer in the burst write is identified by the ST486DX/DX2 by driving BLAST# active. All burst write cycles consist of four data transfers with an address sequence of 0h, 4h, 8h, Ch. For example, if the first address is 1000h, the next three addresses in the burst will be 1004h, 1008h and 100Ch.

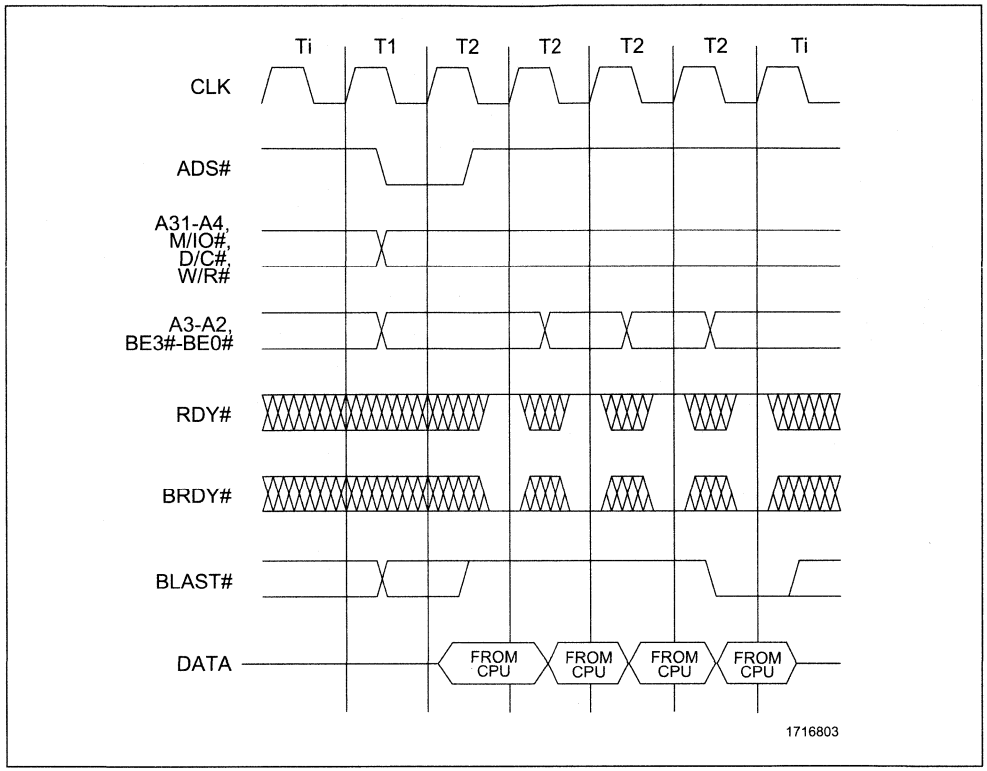


Figure 3 - 8. Burst Write Cycle

3.3.3 SMM Interface

System Management Mode (SMM) uses two ST486DX/DX2 pins, SMI# and SMADS#. The bidirectional SMI# pin is a non-maskable interrupt that is higher priority than the NMI input. SMI# must be active for a least 2 CLK periods to be recognized by the ST486DX/DX2. Once the ST486DX/DX2 recognizes the active SMI# input, the CPU drives the SMI# pin low for the duration of the SMI service routine.

The SMADS# pin outputs the SMM address strobe which indicates that an SMM memory bus cycle is in progress and a valid SMM address is on the address bus. The SMADS# functional timing, output delay times and float delay times are identical to the main memory address strobe (ADS#) timing.

SMI Handshake

The functional timing for the SMI# interrupt is shown in Figure 3-9 (Page 112). Five significant events take place during a ST486DX/DX2 handshake.

(a) The SMI# input pin is driven active (low) by the system logic.
(b) The CPU samples SMI# active on the rising edge of CLK.
(c) One CLK after sampling SMI# active, the CPU switches the SMI# pin to an output and drives SMI# low.
(d) Following execution of the RSM instruction, the CPU drives the SMI# pin high for one CLK indicating completion of the SMI service routine.
(e) The CPU stops driving the SMI# pin high and switches the SMI# pin to an input in preparation for the next SMI interrupt. The system logic is responsible for maintaining the SMI# pin at an inactive (high) level after the pin has been changed to an input.

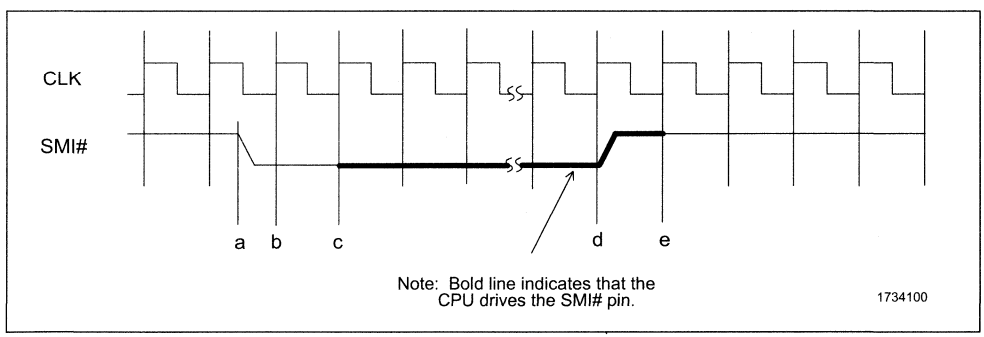


Figure 3 - 9. SMI# Timing

I/O Trapping

The ST486DX/DX2 provides I/O trapping which can be used to facilitate power management of I/O peripherals. When an I/O bus cycle is issued, the I/O address is driven onto the address bus and can be decoded by external logic. If a trap to the SMI handler is required, the SMI# input should be activated at least two CLK edges prior to returning the READY# input for the I/O cycle. The timing for creating an I/O trap via the SMI# input is shown in Figure 3-10. The ST486DX/DX2 immediately traps to the SMI interrupt handler following execution of the I/O instruction, and no other instructions are executed between completion of the I/O instruction and entering the SMI service routine. The I/O trap mechanism is not active during coprocessor accesses.

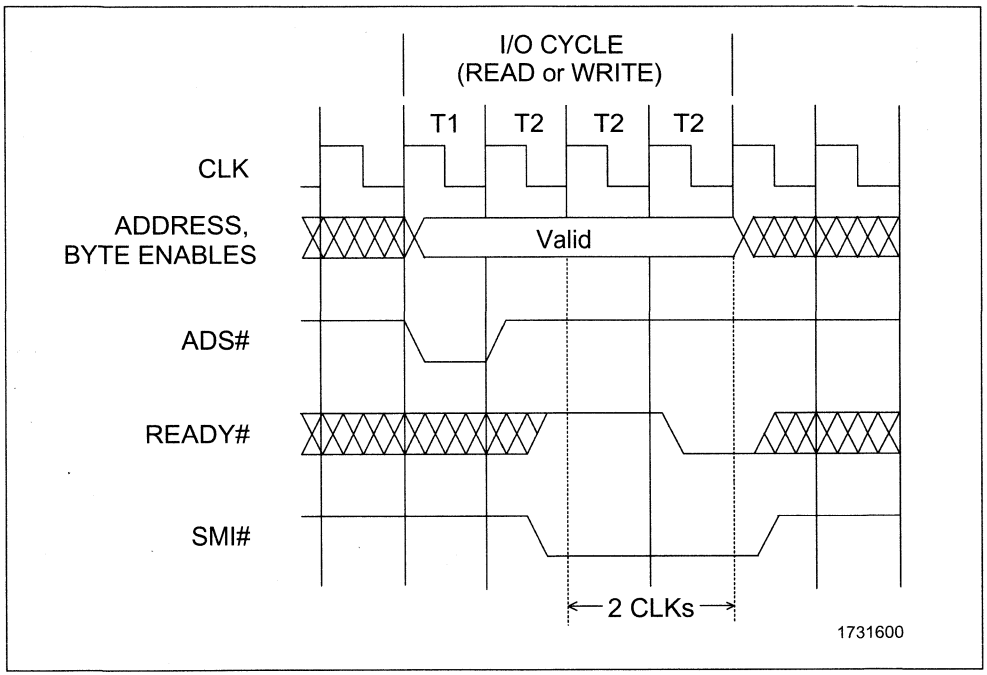


Figure 3 - 10. I/O Trap Timing

3.3.4 Power Management

SUSP# Initiated Suspend Mode

The ST486DX/DX2 enters suspend mode when the SUSP# input is asserted and execution of the current instruction, any pending decoded instructions and associated bus cycles are completed. The SUSPA# output is asserted. The ST486DX/DX2 responds to SUSP# and asserts SUSPA# only if the SUSP bit is set in the CCR2 configuration register.

SUSP# is sampled on the rising edge of CLK. SUSP# must meet specified setup and hold times to be recognized at a particular CLK edge. The time from assertion of SUSP# to activation of SUSPA# varies depending on which instructions were decoded prior to assertion of SUSP#. The minimum time from SUSP# sampled active to SUSPA# asserted is one CLK. As a maximum, the CPU may execute up to two instructions and associated bus cycles prior to asserting SUSPA#. The time required for the CPU to deactivate SUSPA# once SUSP# has been sampled inactive is two CLKs.

If the CPU is in a hold acknowledge state and SUSP# is asserted, the CPU may or may not enter suspend mode depending on the state of the CPU internal execution pipeline. If the CPU is in a SUSP# initiated suspend mode, one occurrence of NMI, INTR and SMI# is stored for execution once suspend mode is exited. The ST486DX/DX2 also recognizes and acknowledges the HOLD input provided the BARB bit in CCR2 is not set.

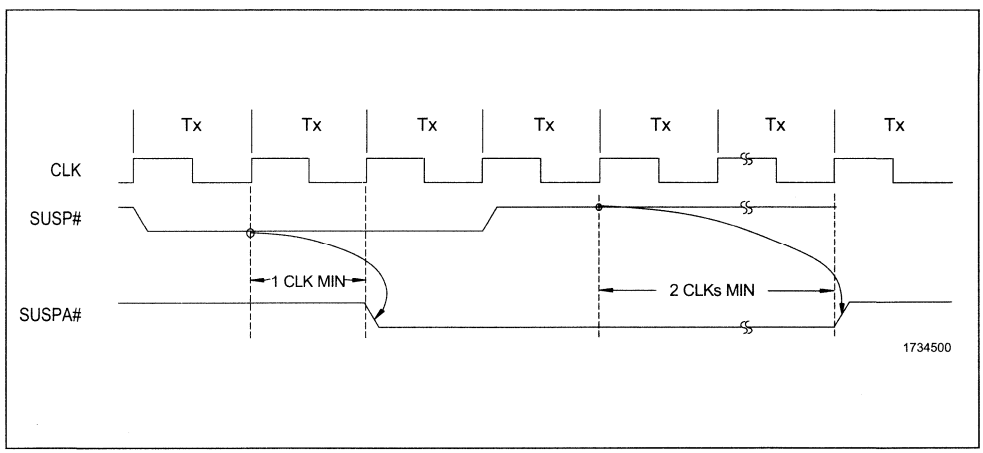


Figure 3 - 11. SUSP# Initiated Suspend Mode

HALT Initiated Suspend Mode

The ST486DX/DX2 also enters suspend mode as a result of executing a HALT instruction if the HALT bit in CCR2 is set. The SUSPA# output is asserted no more than 6 CLKs following RDY# sampled active for the HALT bus cycle as shown in Figure 3-12. Suspend mode is then exited upon recognition of an NMI, an unmasked INTR or an SMI#. SUSPA# is deactivated 6 CLKs after sampling of an active interrupt. The CPU recognizes and acknowledges the HOLD input during suspend mode provided that the BARB bit in CCR2 is not set.

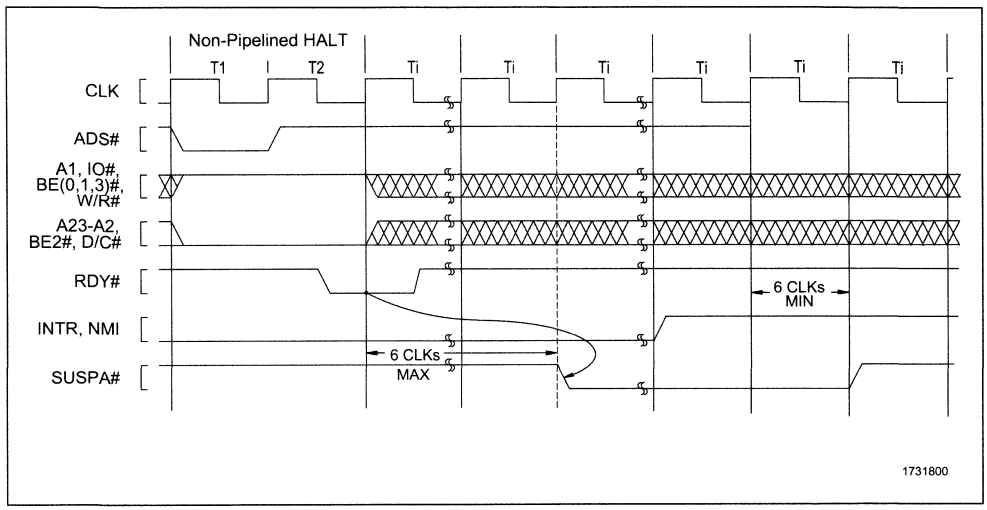


Figure 3 - 12. Halt Initiated Suspend Mode

Stopping the Input Clock

Because the ST486DX/DX2 is a static device, the input clock (CLK) can be stopped and restarted without loss of any internal CPU data. The CLK input can be stopped at either a logic high or logic low state. However, entering suspend mode prior to stopping the CLK dramatically reduces the CPU current requirements. Therefore, the recommended sequence for stopping CLK is to initiate ST486DX/DX2 suspend mode, wait for assertion of SUSPA# by the processor and then stop the input clock.

The ST486DX/DX2 remains suspended until CLK is restarted and suspend mode is exited as described earlier. While the CLK is stopped, the ST486DX/DX2 can no longer sample and respond to any input stimulus including the HOLD, FLUSH#, NMI, INTR and RESET inputs.

Figure 3-13 illustrates the recommended sequence for stopping the CLK using SUSP# to initiate suspend mode. CLK may be started prior to or following negation of the SUSP# input.

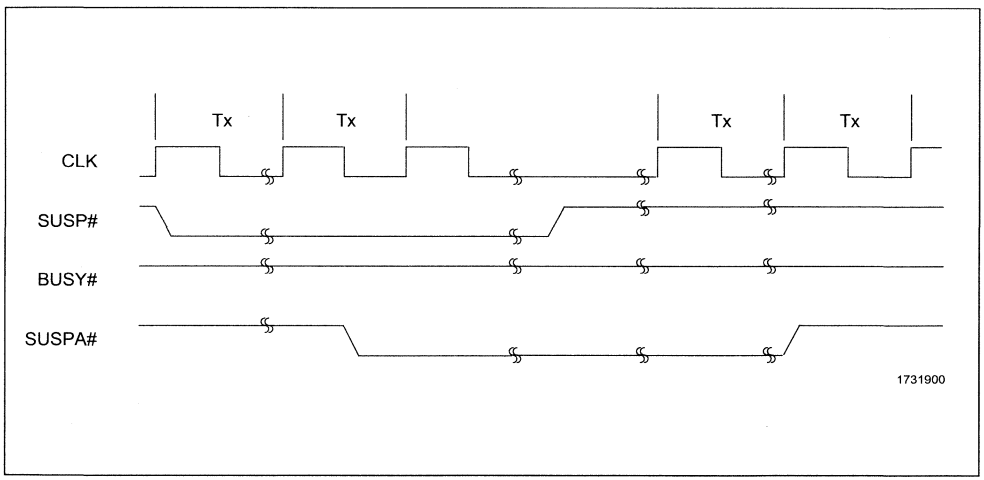


Figure 3 - 13. Stopping CLK During Suspend Mode

ELECTRICAL SPECIFICATIONS

4.0 ELECTRICAL SPECIFICATIONS

Electrical specifications in this chapter are valid for both the ST486DX and the clock-doubled ST486DX2. The ST486DX2 differs from the ST486DX in that the ST486DX2 internal CPU core operates at twice the frequency of the bus interface.

4.1 Electrical Connections

4.1.1 Power and Ground Connections and Decoupling

Due to the high frequency of operation of the ST486DX/DX2, it is necessary to install and test this device using standard high frequency techniques. The high clock frequencies used in the ST486DX/DX2 and its output buffer circuits can cause transient power surges when several output buffers switch output levels simultaneously. These effects can be minimized by filtering the DC power leads with low-inductance decoupling capacitors, using low impedance wiring, and by utilizing all of the VCC and GND pins.

4.1.2 Pull-Up/Pull-Down Resisters

Table 4-1 lists the input pins which are internally connected to pull-up and pull-down resistors. The pull-up resistors are connected to VCC and the pull-down resistors are connected to VSS. When unused, these inputs do not require connection to external pull-up or pull-down resistors. The SUSP# pin is unique in that it is connected to a pull-up resistor only when SUSP# is not asserted.

Table 4 - 1. Pins Connected to Internal Pull-Up and Pull-Down Resistors

SIGNAL	RESISTOR
A20M#	20-kΩ pull-up
AHOLD	20-kΩ pull-down
BOFF#	20-kΩ pull-up
BS16#	20-kΩ pull-up
BS8#	20-kΩ pull-up
BRDY#	20-kΩ pull-up
EADS#	20-kΩ pull-up
FLUSH#	20-kΩ pull-up
IGNNE#	20-kΩ pull-up
INVAL	20-kΩ pull-up
KEN#	20-kΩ pull-up
RDY#	20-kΩ pull-up
UP#	20-kΩ pull-up
SUSP#	20-kΩ pull-up
WM_RST	20-kΩ pull-down

It is recommended that the ADS#, LOCK# and SMI# output pins be connected to pull-up resistors, as indicated in Table 4-2. The external pull-ups guarantee that the signals remain negated during hold acknowledge states.

Table 4 - 2. Pins Requiring External Pull-Up Resistors

SIGNAL	EXTERNAL RESISTOR
ADS#	20-kΩ pull-up
LOCK#	20-kΩ pull-up
SMI#	20-kΩ pull-up

4.1.3 Unused Input Pins

All inputs not used by the system designer and not listed in Table 4-1 (Page 119) should be connected either to ground or to VCC. Connect active-high inputs to ground through a 20 kΩW (±10%) pull-down resistor and active-low inputs to VCC through a 20 kΩW (±10%) pull-up resistor to prevent possible spurious operation.

4.1.4 NC Designated Pins

Pins designated NC should be left disconnected. Connecting an NC pin to a pull-up resistor, pull-down resistor, or an active signal could cause unexpected results and possible circuit malfunctions.

4.2 Absolute Maximum Ratings

The following table lists absolute maximum ratings for the ST486DX/DX2 microprocessors. Stresses beyond those listed under Table 4-3 limits may cause permanent damage to the device. These are stress ratings only and do not imply that operation under any conditions other than those listed under "Recommended Operating Conditions" Table 4-4 (Page 121) is possible. Exposure to conditions beyond Table 4-3 may (1) reduce device reliability and (2) result in premature failure even when there is no immediately apparent sign of failure. Prolonged exposure to conditions at or near the absolute maximum ratings (Table 4-3) may also result in reduced useful life and reliability.

Table 4 - 3. Absolute Maximum Ratings

PARAMETER	ST486DX/DX2		ST486DX/DX2-V		UNITS	NOTES
	MIN	MAX	MIN	MAX		
Case Temperature	-65°	+110°	-65°	+110°	C	Power Applied
Storage Temperature	-65°	+150°	-65°	+150°	C	No Bias
Supply Voltage, VCC	-0.5	6.5	-0.5	4	V	With Respect to V_{SS}
Voltage On Any Pin	-0.5	V_{CC} + 0.5	-0.5	6.0*	V	With Respect to V_{SS}
Input Clamp Current, I_{IK}		10		10	mA	Power Applied
Output Clamp Current, I_{OK}		25		25	mA	Power Applied

*Note: The maximum "Voltage On Any Pin" for ST486DX/DX2-V devices with a stepping ID of 0xh is V_{CC}+0.5 volts; for devices with stepping IDs of 1xh and above the maximum voltage is 6.0 volts, as shown. (The stepping ID for a ST486DX/DX2-V CPU can be found by reading the DIR1 configuration register.)

4.3 Recommended Operating Conditions

Table 4-4 presents the recommended operating conditions for the ST486DX/DX2 device.

Table 4 - 4. Recommended Operating Conditions

PARAMETER	ST486DX/DX2		ST486DX/DX2-V		UNITS	NOTES
	MIN	MAX	MIN	MAX		
T_C Case Temperature	0°	+85°	0°	+85°	C	Power Applied
V_{CC} Supply Voltage	4.75	5.25	3	3.6	V	With Respect to Vss
V_{IH} High Level Input	2	V_{CC}+0.3	2	5.5	V	See note.
V_{IL} Low Level Input	-0.3	0.8	-0.3	0.6	V	
I_{OH} Output Current (High)		-1		-1	mA	$V_{OH}=V_{OH(MIN)}$
I_{OL} Output Current (Low)		5		3	mA	$V_{OL}=V_{OL(MAX)}$

Note: V_{IH} specification shown for the ST486DX/DX2-V applies to devices with stepping IDs of 1xh and greater (DIR1 configuration register). ST486DX/DX2-V devices with a stepping ID of 0xh require a $V_{IH}(max) = V_{CC}+0.3$ volts.

4.4 DC Characteristics

Table 4 - 5. DC Characteristics (at Recommended Operating Conditions)

PARAMETER	ST486DX/DX2		ST486DX/DX2-V		UNITS	NOTES
	MIN	MAX	MIN	MAX		
V_{OL} Output Low Voltage I_{OL} = 5 mA		0.45		0.35	V	
V_{OH} Output High Voltage I_{OH} = -1 mA	2.4		2.4		V	
I_{LI} Input Leakage Current For all pins except those listed in Table 4-1.		±15		±15	µA	$0<V_{IN}<V_{CC}$
I_{IH} Input Leakage Current For all pins with internal pull-downs.		200		200	µA	V_{IH} = 2.4 V See Table 4-1.
I_{IL} Input Leakage Current For all pins with internal pull-ups		-400		-400	µA	V_{IL} = 0.45 V See Table 4-1.
I_{CC} Active I_{CC} 33 MHz 40 MHz 50 MHz 66 MHz 80 MHz	Typical: 610 685 765 860	 925 1025 1170 1325	Typical: 420 450 495 560 630	 640 680 750 850 950	µA	Note 1
I_{CCSM} Suspend Mode I_{CC} 33 MHz 40 MHz 50 MHz 66 MHz 80 MHz	Typical: 12.5 13.5 15.5 18	 24.5 27 31 35	Typical: 9 10 12 14 16	 20 23 26 30 34	µA	Note 1, 3
I_{CCSS} Standby I_{CC} 0 MHz (Suspended/CLK Stopped)	Typical: 0.45	 1.1	Typical: 0.45	 1.1	µA	Note 4
C_{IN} Input Capacitance		20		20	pF	f_c = 1 MHz (Note 2)
C_{OUT} Output or I/O Capacitance		20		20	pF	f_c = 1 MHz (Note 2)
C_{CLK} CLK Capacitance		20		20	pF	f_c = 1 MHz (Note 2)

Notes:
1. MHz ratings refer to internal clock frequency.
2. Not 100% tested.
3. All inputs at 0.4 or VCC - 0.4 (CMOS levels). All inputs held static except clock and all outputs unloaded (static IOUT = 0 mA). Specification also valid for UP# = 0.
4. All inputs at 0.4 or VCC - 0.4 (CMOS levels). All inputs held static and all outputs unloaded (static IOUT = 0 mA).

4.5 AC Characteristics

Tables 4-7 through 4-12 (Pages 125 through 130) list the AC characteristics including output delays, input setup requirements, input hold requirements and output float delays. These measurements are based on the measurement points identified in Figure 4-1 (Page 124) and Figure 4-2 (Page 124). The rising clock edge reference level VREF , and other reference levels are shown in Table 4-6 below for the ST486DX/DX2. Input or output signals must cross these levels during testing.

Figure 4-1 (Page 124) shows output delay (A and B) and input setup and hold times (C and D). Input setup and hold times (C and D) are specified minimums, defining the smallest acceptable sampling window a synchronous input signal must bc stable for correct operation.

Table 4 - 6. Drive Level and Measurement Points for Switching Characteristics

SYMBOL	ST486DX/DX2	ST486DX/DX2-V	UNITS
V_{REF}	1.5	1.5	V
V_{IHD}	3	2.3	V
V_{ILD}	0	0	V
Note: Refer to Figure 4-1.			

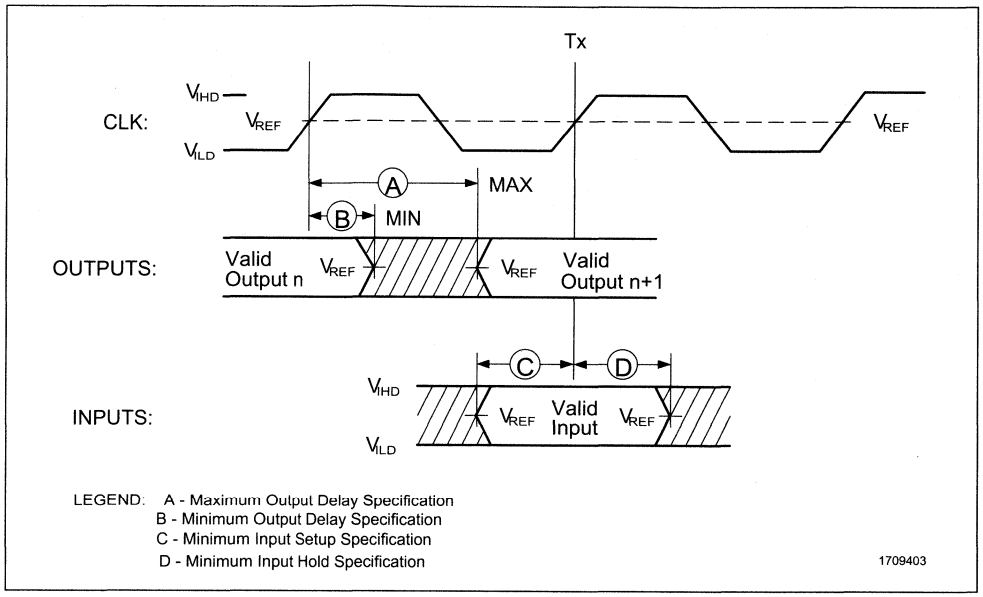


Figure 4 - 1. Drive Level and Measurement Points for Switching Characteristics

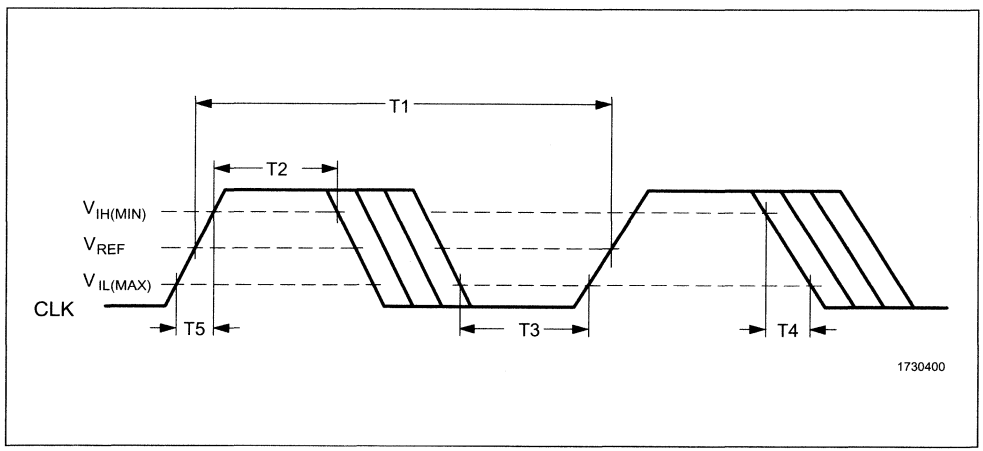


Figure 4 - 2. CLK Timing Measurement Points

Table 4 - 7. AC Characteristics for ST486DX2-50

V_{CC} = 5.0 V ± 5%, T_{CASE} = 0° to 85° C, C_L = 50pF
External CLK = 25 MHz (Max.)

SYMBOL	PARAMETERS	MIN (ns)	MAX (ns)	FIGURE	NOTES
T1	CLK Period	40		4-2	
T2	CLK High Time	14		4-2	At 2 V
T3	CLK Low Time	14		4-2	$V_{IL(MAX)}$
T4	CLK Fall Time		4	4-2	2 V to $V_{IL(MAX)}$
T5	CLK Rise Time		4	4-2	$V_{IL(MAX)}$ to 2 V
T6	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, HLDA, FERR#, LOCK#, M/IO#, PCD, PWT, W/R# Valid Delay	3	19	4-6	
T6a	SMADS#, SMI# Valid Delay	3	19	4-6	
T7	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R# Float Delay		28	4-7	Note 1
T7a	SMADS#, SMI# Float Delay		28	4-7	Note 1
T8	PCHK# Valid Delay	3	24	4-5	
T8a	BLAST#, PLOCK# Valid Delay	3	24	4-6	
T8b	HITM#, RPLSET(1-0), RPLVAL#, SUSPA# Valid Delay	3	24	4-6	
T9	BLAST#, PLOCK# Float Delay		28	4-7	Note 1
T9a	RPLSET(1-0), RPLVAL# Float Delay		28	4-7	Note 1
T10	D31-D0, DP3-DP0 Write Data Valid Delay	3	20	4-6	
T11	D31-D0, DP3-DP0 Write Data Float Delay		28	4-7	Note 1
T12	EADS# Setup Time	8		4-3	
T12a	INVAL Setup Time	8		4-3	
T13	EADS# Hold Time	3		4-3	
T13a	INVAL Hold Time	3		4-3	
T14	BS16#, BS8#, KEN# Setup Time	8		4-3	
T15	BS16#, BS8#, KEN# Hold Time	3		4-3	
T16	BRDY#, RDY# Setup Time	8		4-4	
T17	BRDY#, RDY# Hold Time	3		4-4	
T18	AHOLD, HOLD Setup Time	10		4-3	
T18a	BOFF# Setup Time	10		4-3	
T19	AHOLD, BOFF#, HOLD Hold Time	3		4-3	
T20	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESET Setup Time	10		4-3	
T20a	SMI#, SUSP#, WM_RST Setup Time	10		4-3	
T21	A20M#, FLUSH#, INTR, IGNNE#, NMI, RESET Hold Time	3		4-3	
T21a	SMI#, SUSP#, WM_RST Hold Time	3		4-3	
T22	A31-A4, D31-D0, DP3-DP0 Read Setup Time	5		4-3, 4-4	
T23	A31-A4, D31-D0, DP3-DP0 Read Hold Time	3		4-3, 4-4	

Note 1: Not 100% tested.

Table 4 - 8. AC Characteristics for ST486DX-33, ST486DX2-66

V_{CC} = 5.0 V ± 5%, T_{CASE} = 0° to 85° C, C_L = 50pF
External CLK = 33 MHz (Max.)

SYMBOL	PARAMETERS	MIN (ns)	MAX(ns)	FIGURE	NOTES
T1	CLK Period	30		4-2	
T2	CLK High Time	11		4-2	At 2 V
T3	CLK Low Time	11		4-2	$V_{IL(MAX)}$
T4	CLK Fall Time		3	4-2	2 V to $V_{IL(MAX)}$
T5	CLK Rise Time		3	4-2	$V_{IL(MAX)}$ to 2 V
T6	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, FERR#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R#Valid Delay	3	16	4-6	
T6a	SMADS#, SMI# Valid Delay	3	16	4-6	
T7	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R# Float Delay		20	4-7	Note 1
T7a	SMADS#, SMI# Float Delay		20	4-7	Note 1
T8	PCHK# Valid Delay	3	22	4-5	
T8a	BLAST#, PLOCK# Valid Delay	3	20	4-6	
T8b	HITM#, RPLSET(1-0), RPLVAL#, SUSPA# Valid Delay	3	20	4-6	
T9	BLAST#, PLOCK# Float Delay		20	4-7	Note 1
T9a	RPLSET(1-0), RPLVAL# Float Delay		20	4-7	Note 1
T10	D31-D0, DP3-DP0 Write Data Valid Delay	3	18	4-6	
T11	D31-D0, DP3-DP0 Write Data Float Delay		20	4-7	Note 1
T12	EADS# Setup Time	5		4-3	
T12a	INVAL Setup Time	5		4-3	
T13	EADS# Hold Time	3		4-3	
T13a	INVAL Hold Time	3		4-3	
T14	BS16#, BS8#, KEN# Setup Time	5		4-3	
T15	BS16#, BS8#, KEN# Hold Time	3		4-3	
T16	BRDY#, RDY# Setup Time	5		4-4	
T17	BRDY#, RDY# Hold Time	3		4-4	
T18	AHOLD, HOLD Setup Time	6		4-3	
T18a	BOFF# Setup Time	8		4-3	
T19	AHOLD, BOFF#, HOLD Hold Time	3		4-3	
T20	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESET Setup Time	5		4-3	
T20a	SMI#, SUSP#, WM_RST Setup Time	5		4-3	
T21	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESET Hold Time	3		4-3	
T21a	SMI#, SUSP#, WM_RST Hold Time	3		4-3	
T22	A31-A4, D31-D0, DP3-DP0 Read Setup Time	5		4-3, 4-4	
T23	A31-A4, D31-D0, DP3-DP0 Read Hold Time	3		4-3, 4-4	
Note 1: Not 100% tested.					

Table 4 - 9. AC Characteristics for ST486DX-40

V_{CC} = 5.0 V ± 5%, T_{CASE} = 0° to 85° C, C_L = 50pF
External CLK = 40 MHz (Max.)

SYMBOL	PARAMETERS	MIN (ns)	MAX (ns)	FIGURE	NOTES
T1	CLK Period	25		4-2	
T2	CLK High Time	9		4-2	At 2 V
T3	CLK Low Time	9		4-2	$V_{IL(MAX)}$
T4	CLK Fall Time		3	4-2	2 V to $V_{IL(MAX)}$
T5	CLK Rise Time		3	4-2	$V_{IL(MAX)}$ to 2 V
T6	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, FERR#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R# Valid Delay	3	14	4-6	
T6a	SMADS#, SMI# Valid Delay	3	14	4-6	
T7	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, FERR#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R# Float Delay		19	4-7	Note 1
T7a	SMADS#, SMI# Float Delay		19	4-7	Note 1
T8	PCHK# Valid Delay	3	18	4-5	
T8a	BLAST#, PLOCK# Valid Delay	3	16	4-6	
T8b	HITM#, RPLSET(1-0), RPLVAL#, SUSPA# Valid Delay	3	16	4-6	
T9	BLAST#, PLOCK# Float Delay		19	4-7	Note 1
T9a	RPLSET(1-0), RPLVAL# Float Delay		19	4-7	Note 1
T10	D31-D0, DP3-DP0 Write Data Valid Delay	3	16	4-6	
T11	D31-D0, DP3-DP0 Write Data Float Delay		19	4-7	Note 1
T12	EADS# Setup Time	5		4-3	
T12a	INVAL Setup Time	5		4-3	
T13	EADS# Hold Time	3		4-3	
T13a	INVAL Hold Time	3		4-3	
T14	BS16#, BS8#, KEN# Setup Time	5		4-3	
T15	BS16#, BS8#, KEN# Hold Time	3		4-3	
T16	BRDY#, RDY# Setup Time	5		4-4	
T17	BRDY#, RDY# Hold Time	3		4-4	
T18	AHOLD, HOLD Setup Time	5		4-3	
T18a	BOFF# Setup Time	6		4-3	
T19	AHOLD, BOFF#, HOLD Hold Time	3		4-3	
T20	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESET Setup Time	5		4-3	
T20a	SMI#, SUSP#, WM_RST Setup Time	5		4-3	
T21	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESET Hold Time	3		4-3	
T21a	SMI#, SUSP#, WM_RST Hold Time	3		4-3	
T22	A31-A4, D31-D0, DP3-DP0 Read Setup Time	5		4-3, 4-4	
T23	A31-A4, D31-D0, DP3-DP0 Read Hold Time	3		4-3, 4-4	
Note 1: Not 100% tested.					

Table 4 - 10. AC Characteristics for ST486DX-50

V_{CC} = 5.0 V ± 5%, T_{CASE} = 0° to 85° C, C_L = 50pF
External CLK = 50 MHz (Max.)

SYMBOL	PARAMETERS	MIN (ns)	MAX (ns)	FIGURE	NOTES
T1	CLK Period	20		4-2	
T2	CLK High Time	7		4-2	At 2 V
T3	CLK Low Time	7		4-2	$V_{IL(MAX)}$
T4	CLK Fall Time		2	4-2	2 V to $V_{IL(MAX)}$
T5	CLK Rise Time		2	4-2	$V_{IL(MAX)}$ to 2 V
T6	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, FERR#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R# Valid Delay	3	12	4-6	
T6a	SMADS#, SMI# Valid Delay	3	12	4-6	
T7	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R# Float Delay		18	4-7	Note 1
T7a	SMADS#, SMI# Float Delay		18	4-7	Note 1
T8	PCHK# Valid Delay	3	14	4-5	
T8a	BLAST#, PLOCK# Valid Delay	3	12	4-6	
T8b	HITM#, RPLSET(1-0), RPLVAL#, SUSPA# Valid Delay	3	14	4-6	
T9	BLAST#, PLOCK# Float Delay		18	4-7	Note 1
T9a	RPLSET(1-0), RPLVAL# Float Delay		18	4-7	Note 1
T10	D31-D0, DP3-DP0 Write Data Valid Delay	3	12	4-6	
T11	D31-D0, DP3-DP0 Write Data Float Delay		18	4-7	Note 1
T12	EADS# Setup Time	5		4-3	
T12a	INVAL Setup Time	5		4-3	
T13	EADS# Hold Time	2		4-3	
T13a	INVAL Hold Time	2		4-3	
T14	BS16#, BS8#, KEN# Setup Time	5		4-3	
T15	BS16#, BS8#, KEN# Hold Time	2		4-3	
T16	BRDY#, RDY# Setup Time	5		4-4	
T17	BRDY#, RDY# Hold Time	2		4-4	
T18	AHOLD, HOLD Setup Time	5		4-3	
T18a	BOFF# Setup Time	5		4-3	
T19	AHOLD, BOFF#, HOLD Hold Time	2		4-3	
T20	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESET Setup Time	5		4-3	
T20a	SMI#, SUSP#, WM_RST Setup Time	5		4-3	
T21	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESETHold Time	2		4-3	
T21a	SMI#, SUSP#, WM_RST Hold Time	2		4-3	
T22	A31-A4, D31-D0, DP3-DP0 Read Setup Time	4		4-3, 4-4	
T23	A31-A4, D31-D0, DP3-DP0 Read Hold Time	2		4-3, 4-4	
Note 1: Not 100% tested.					

Table 4 - 11. AC Characteristics for ST486DX-V50

V_{CC} = 3.0 to 3.6 V, T_{CASE} = 0° to 85° C, C_L = 50pF
External CLK = 25 MHz (Max.)

SYMBOL	PARAMETERS	MIN (ns)	MAX (ns)	FIGURE	NOTES
T1	CLK Period	40		4-2	
T2	CLK High Time	14		4-2	At 2 V
T3	CLK Low Time	14		4-2	$V_{IL(MAX)}$
T4	CLK Fall Time		4	4-2	2 V to $V_{IL(MAX)}$
T5	CLK Rise Time		4	4-2	$V_{IL(MAX)}$ to 2 V
T6	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, HLDA, FERR#, LOCK#, M/IO#, PCD, PWT, W/R# Valid Delay	3	19	4-6	
T6a	SMADS#, SMI# Valid Delay	3	19	4-6	
T7	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R# Float Delay		28	4-7	Note 1
T7a	SMADS#, SMI# Float Delay		28	4-7	Note 1
T8	PCHK# Valid Delay	3	24	4-5	
T8a	BLAST#, PLOCK# Valid Delay	3	24	4-6	
T8b	HITM#, RPLSET(1-0), RPLVAL#, SUSPA# Valid Delay	3	24	4-6	
T9	BLAST#, PLOCK# Float Delay		28	4-7	Note 1
T9a	RPLSET(1-0), RPLVAL# Float Delay		28	4-7	Note 1
T10	D31-D0, DP3-DP0 Write Data Valid Delay	3	20	4-6	
T11	D31-D0, DP3-DP0 Write Data Float Delay		28	4-7	Note 1
T12	EADS# Setup Time	8		4-3	
T12a	INVAL Setup Time	8		4-3	
T13	EADS# Hold Time	3		4-3	
T13a	INVAL Hold Time	3		4-3	
T14	BS16#, BS8#, KEN# Setup Time	8		4-3	
T15	BS16#, BS8#, KEN# Hold Time	3		4-3	
T16	BRDY#, RDY# Setup Time	8		4-4	
T17	BRDY#, RDY# Hold Time	3		4-4	
T18	AHOLD, HOLD Setup Time	10		4-3	
T18a	BOFF# Setup Time	10		4-3	
T19	AHOLD, BOFF#, HOLD Hold Time	3		4-3	
T20	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESET Setup Time	10		4-3	
T20a	SMI#, SUSP#, WM_RST Setup Time	10		4-3	
T21	A20M#, FLUSH#, INTR, IGNNE#, NMI, RESET Hold Time	3		4-3	
T21a	SMI#, SUSP#, WM_RST Hold Time	3		4-3	
T22	A31-A4, D31-D0, DP3-DP0 Read Setup Time	6		4-3, 4-4	
T23	A31-A4, D31-D0, DP3-DP0 Read Hold Time	3		4-3, 4-4	
Note 1: Not 100% tested.					

Table 4 - 12. AC Characteristics for ST486DX-V33, ST486DX-V66

V_{CC} = 3.0 to 3.6 V, T_{CASE} = 0° to 85° C, C_L = 50pF
External CLK = 33 MHz (Max.)

SYMBOL	PARAMETERS	MIN (ns)	MAX (ns)	FIGURE	NOTES
T1	CLK Period	30		4-2	
T2	CLK High Time	11		4-2	At 2 V
T3	CLK Low Time	11		4-2	$V_{IL(MAX)}$
T4	CLK Fall Time		3	4-2	2 V to $V_{IL(MAX)}$
T5	CLK Rise Time		3	4-2	$V_{IL(MAX)}$ to 2 V
T6	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, FERR#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R# Valid Delay	3	16	4-6	
T6a	SMADS#, SMI# Valid Delay	3	16	4-6	
T7	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R# Float Delay		20	4-7	Note 1
T7a	SMADS#, SMI# Float Delay		20	4-7	Note 1
T8	PCHK# Valid Delay	3	22	4-5	
T8a	BLAST#, PLOCK# Valid Delay	3	20	4-6	
T8b	HITM#, RPLSET(1-0), RPLVAL#, SUSPA# Valid Delay	3	20	4-6	
T9	BLAST#, PLOCK# Float Delay		20	4-7	Note 1
T9a	RPLSET(1-0), RPLVAL# Float Delay		20	4-7	Note 1
T10	D31-D0, DP3-DP0 Write Data Valid Delay	3	19	4-6	
T11	D31-D0, DP3-DP0 Write Data Float Delay		20	4-7	Note 1
T12	EADS# Setup Time	6		4-3	
T12a	INVAL Setup Time	6		4-3	
T13	EADS# Hold Time	3		4-3	
T13a	INVAL Hold Time	3		4-3	
T14	BS16#, BS8#, KEN# Setup Time	6		4-3	
T15	BS16#, BS8#, KEN# Hold Time	3		4-3	
T16	BRDY#, RDY# Setup Time	6		4-4	
T17	BRDY#, RDY# Hold Time	3		4-4	
T18	AHOLD, HOLD Setup Time	6		4-3	
T18a	BOFF# Setup Time	9		4-3	
T19	AHOLD, BOFF#, HOLD Hold Time	3		4-3	
T20	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESET Setup Time	6		4-3	
T20a	SMI#, SUSP#, WM_RST Setup Time	6		4-3	
T21	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESET Hold Time	3		4-3	
T21a	SMI#, SUSP#, WM_RST Hold Time	3		4-3	
T22	A31-A4, D31-D0, DP3-DP0 Read Setup Time	6		4-3, 4-4	
T23	A31-A4, D31-D0, DP3-DP0 Read Hold Time	3		4-3, 4-4	
Note 1: Not 100% tested.					

Table 4 - 13. Characteristics for ST486DX-V40, ST486DX-V80

V_{CC} = 3.0 to 3.6 V, T_{CASE} = 0° to 85° C, C_L = 50pF
External CLK = 40 MHz (Max.)

SYMBOL	PARAMETERS	MIN (ns)	MAX (ns)	FIGURE	NOTES
T1	CLK Period	25		4-2	
T2	CLK High Time	9		4-2	At 2 V
T3	CLK Low Time	9		4-2	$V_{IL(MAX)}$
T4	CLK Fall Time		3	4-2	2 V to $V_{IL(MAX)}$
T5	CLK Rise Time		3	4-2	$V_{IL(MAX)}$ to 2 V
T6	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, FERR#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R# Valid Delay	3	14	4-6	
T6a	SMADS#, SMI# Valid Delay	3	14	4-6	
T7	A31-A2, ADS#, BE3#-BE0#, BREQ, D/C#, HLDA, LOCK#, M/IO#, PCD, PWT, W/R# Float Delay		19	4-7	Note 1
T7a	SMADS#, SMI# Float Delay		19	4-7	Note 1
T8	PCHK# Valid Delay	3	18	4-5	
T8a	BLAST#, PLOCK# Valid Delay	3	16	4-6	
T8b	HITM#, RPLSET(1-0), RPLVAL#, SUSPA# Valid Delay	3	16	4-6	
T9	BLAST#, PLOCK# Float Delay		16	4-7	Note 1
T9a	RPLSET(1-0), RPLVAL# Float Delay		16	4-7	Note 1
T10	D31-D0, DP3-DP0 Write Data Valid Delay	3	17	4-6	
T11	D31-D0, DP3-DP0 Write Data Float Delay		19	4-7	Note 1
T12	EADS# Setup Time	6		4-3	
T12a	INVAL Setup Time	6		4-3	
T13	EADS# Hold Time	3		4-3	
T13a	INVAL Hold Time	3		4-3	
T14	BS16#, BS8#, KEN# Setup Time	6		4-3	
T15	BS16#, BS8#, KEN# Hold Time	3		4-3	
T16	BRDY#, RDY# Setup Time	6		4-4	
T17	BRDY#, RDY# Hold Time	3		4-4	
T18	AHOLD, HOLD Setup Time	6		4-3	
T18a	BOFF# Setup Time	7		4-3	
T19	AHOLD, BOFF#, HOLD Hold Time	3		4-3	
T20	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESET Setup Time	6		4-3	
T20a	SMI#, SUSP#, WM_RST Setup Time	6		4-3	
T21	A20M#, FLUSH#, IGNNE#, INTR, NMI, RESET Hold Time	3		4-3	
T21a	SMI#, SUSP#, WM_RST Hold Time	3		4-3	
T22	A31-A4, D31-D0, DP3-DP0 Read Setup Time	6		4-3, 4-4	
T23	A31-A4, D31-D0, DP3-DP0 Read Hold Time	3		4-3, 4-4	
Note 1: Not 100% tested.					

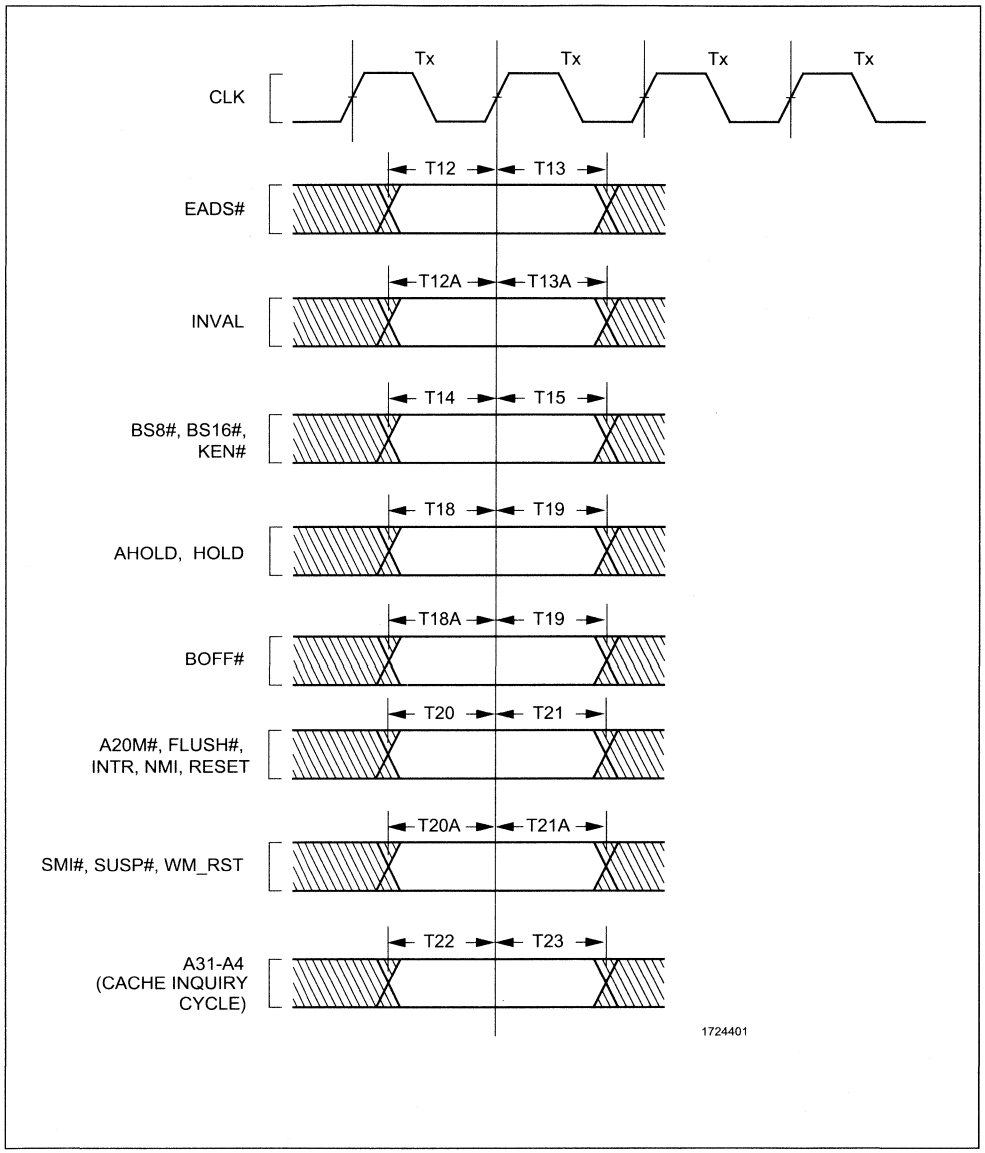


Figure 4 - 3. Input Setup and Hold Timing

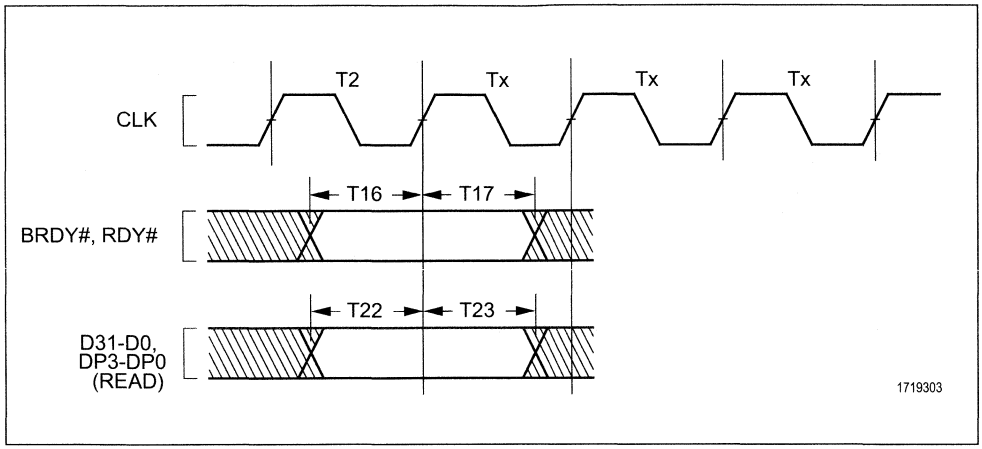


Figure 4 - 4. Input Setup and Hold Timing (Continued)

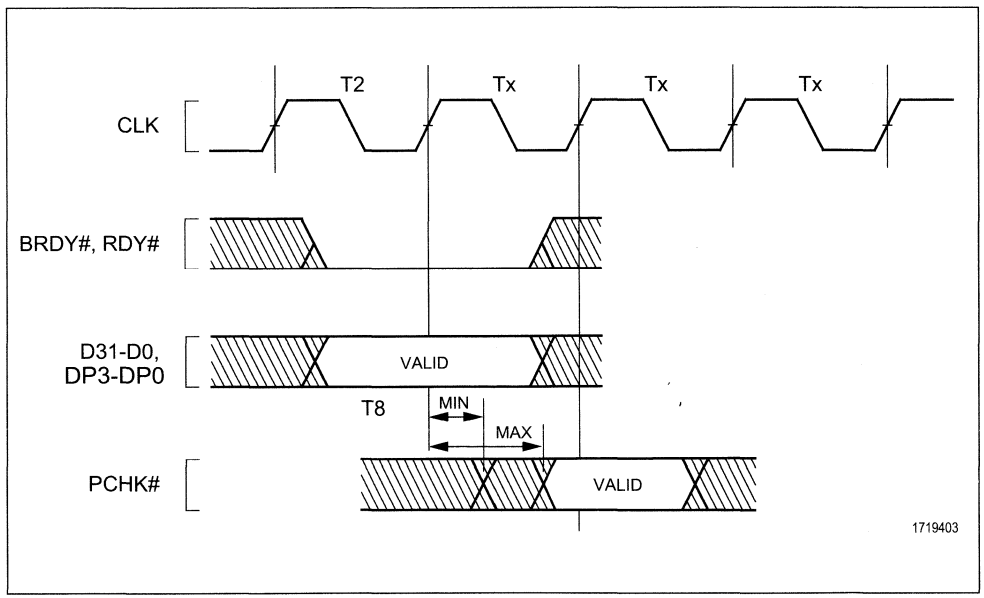


Figure 4 - 5. PCHK# Valid Delay Timing

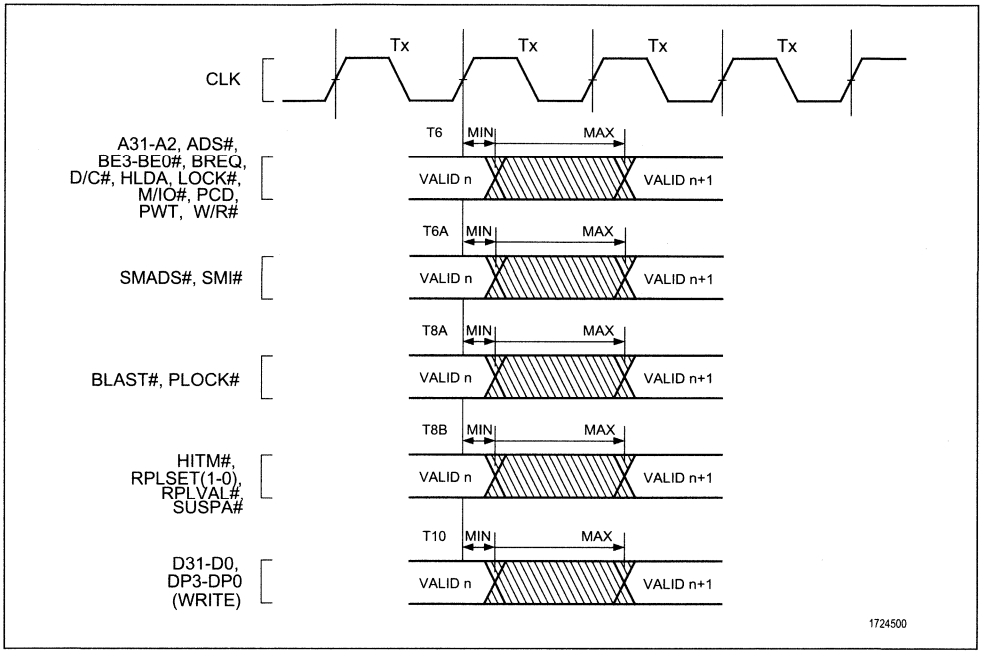


Figure 4 - 6. Output Valid Delay Timing

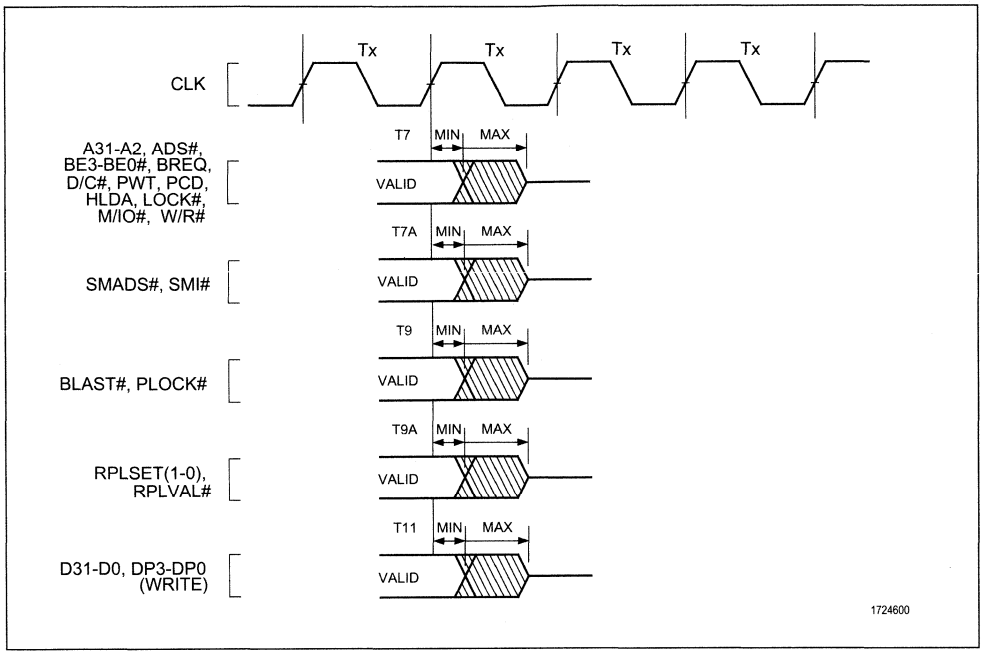


Figure 4 - 7. Maximum Float Delay Timing

MECHANICAL SPECIFICATIONS

5.0 MECHANICAL SPECIFICATIONS

5.1 168-Pin PGA Package

The pin assignments for the ST486DX/DX2 168-pin PGA package are shown in Figure 5-1. The pins are listed by signal name and pin number in Tables 5-2 and 5-3. Dimensions are shown in Figure 5-2 and Table 5-4.

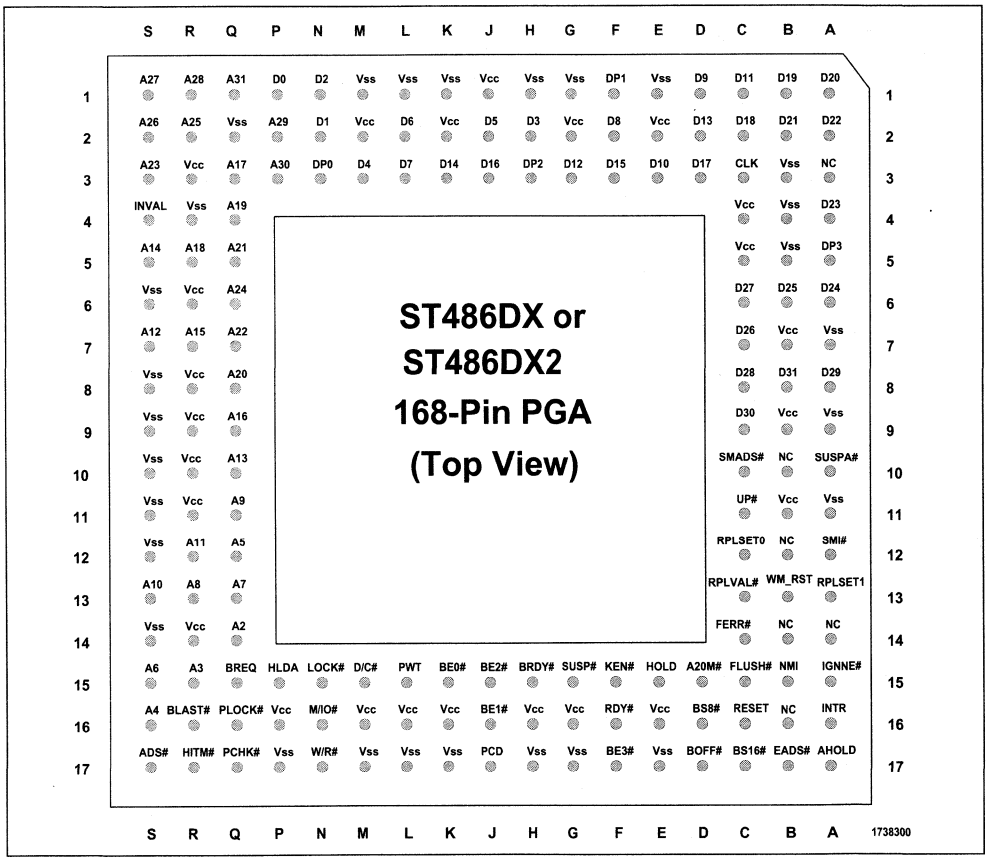


Figure 5 - 1. 168-Pin PGA Package Pin Assignments

Table 5 - 2. 168-Pin PGA Package Pin Numbers Sorted by Signal Name

Signal Name	Pin No.	Signal Name	Pin No.	Signal Name	Pin No.	Signal Name	Pin No.	Signal Name	Pin No.	Signal Name	Pin No.
A2	Q14	A29	P2	D11	C1	HITM#	R17	SUSP#	G15	VSS	A11
A3	R15	A30	P3	D12	G3	HLDA	P15	SUSPA#	A10	VSS	B3
A4	S16	A31	Q1	D13	D2	HOLD	E15	UP#	C11	VSS	B4
A5	Q12	ADS#	S17	D14	K3	IGNNE#	A15	VCC	B7	VSS	B5
A6	S15	AHOLD	A17	D15	F3	INTR	A16	VCC	B9	VSS	E1
A7	Q13	BE0#	K15	D16	J3	INVAL	S4	VCC	B11	VSS	E17
A8	R13	BE1#	J16	D17	D3	KEN#	F15	VCC	C4	VSS	G1
A9	Q11	BE2#	J15	D18	C2	LOCK#	N15	VCC	C5	VSS	G17
A10	S13	BE3#	F17	D19	B1	M/IO#	N16	VCC	E2	VSS	H1
A11	R12	BLAST#	R16	D20	A1	NC	A3	VCC	E16	VSS	H17
A12	S7	BOFF#	D17	D21	B2	NC	A14	VCC	G2	VSS	K1
A13	Q10	BRDY#	H15	D22	A2	NC	B10	VCC	G16	VSS	K17
A14	S5	BREQ	Q15	D23	A4	NC	B12	VCC	H16	VSS	L1
A15	R7	BS8#	D16	D24	A6	NC	B14	VCC	K2	VSS	L17
A16	Q9	BS16#	C17	D25	B6	NC	B16	VCC	K16	VSS	M1
A17	Q3	CLK	C3	D26	C7	NC	J1	VCC	L16	VSS	M17
A18	R5	D/C#	M15	D27	C6	NMI	B15	VCC	M2	VSS	P17
A19	Q4	D0	P1	D28	C8	PCD	J17	VCC	M16	VSS	Q2
A20	Q8	D1	N2	D29	A8	PCHK#	Q17	VCC	P16	VSS	R4
A20M#	D15	D2	N1	D30	C9	PLOCK#	Q16	VCC	R3	VSS	S6
A21	Q5	D3	H2	D31	B8	PWT	L15	VCC	R6	VSS	S8
A22	Q7	D4	M3	DP0	N3	RDY#	F16	VCC	R8	VSS	S9
A23	S3	D5	J2	DP1	F1	RESET	C16	VCC	R9	VSS	S10
A24	Q6	D6	L2	DP2	H3	RPLSET0	C12	VCC	R10	VSS	S11
A25	R2	D7	L3	DP3	A5	RPLSET1	A13	VCC	R11	VSS	S12
A26	S2	D8	F2	EADS#	B17	RPLVAL#	C13	VCC	R14	VSS	S14
A27	S1	D9	D1	FERR#	C14	SMADS#	C10	VSS	A7	W/R#	N17
A28	R1	D10	E3	FLUSH#	C15	SMI#	A12	VSS	A9	WM_RST	B13

Table 5 - 3. 168-Pin PGA Package Signal Names Sorted by Pin Number

Pin No.	Signal Name	Pin No.	Signal Name	Pin No.	Signal Name	Pin No.	Signal Name	Pin No.	Signal Name	Pin No.	Signal Name
A1	D20	B12	NC	D17	BOFF#	J15	BE2#	P2	A29	R7	A15
A2	D22	B13	WM_RST	E1	VSS	J16	BE1#	P3	A30	R8	VCC
A3	NC	B14	NC	E2	VCC	J17	PCD	P15	HLDA	R9	VCC
A4	D23	B15	NMI	E3	D10	K1	VSS	P16	VCC	R10	VCC
A5	DP3	B16	NC	E15	HOLD	K2	VCC	P17	VSS	R11	VCC
A6	D24	B17	EADS#	E16	VCC	K3	D14	Q1	A31	R12	A11
A7	VSS	C1	D11	E17	VSS	K15	BE0#	Q2	VSS	R13	A8
A8	D29	C2	D18	F1	DP1	K16	VCC	Q3	A17	R14	VCC
A9	VSS	C3	CLK	F2	D8	K17	VSS	Q4	A19	R15	A3
A10	SUSPA#	C4	VCC	F3	D15	L1	VSS	Q5	A21	R16	BLAST#
A11	VSS	C5	VCC	F15	KEN#	L2	D6	Q6	A24	R17	HITM#
A12	SMI#	C6	D27	F16	RDY#	L3	D7	Q7	A22	S1	A27
A13	RPLSET1	C7	D26	F17	BE3#	L15	PWT	Q8	A20	S2	A26
A14	NC	C8	D28	G1	VSS	L16	VCC	Q9	A16	S3	A23
A15	IGNNE#	C9	D30	G2	VCC	L17	VSS	Q10	A13	S4	INVAL
A16	INTR	C10	SMADS#	G3	D12	M1	VSS	Q11	A9	S5	A14
A17	AHOLD	C11	UP#	G15	SUSP#	M2	VCC	Q12	A5	S6	VSS
B1	D19	C12	RPLSET0	G16	VCC	M3	D4	Q13	A7	S7	A12
B2	D21	C13	RPLVAL#	G17	VSS	M15	D/C#	Q14	A2	S8	VSS
B3	VSS	C14	FERR#	H1	VSS	M16	VCC	Q15	BREQ	S9	VSS
B4	VSS	C15	FLUSH#	H2	D3	M17	VSS	Q16	PLOCK#	S10	VSS
B5	VSS	C16	RESET	H3	DP2	N1	D2	Q17	PCHK#	S11	VSS
B6	D25	C17	BS16#	H15	BRDY#	N2	D1	R1	A28	S12	VSS
B7	VCC	D1	D9	H16	VCC	N3	DP0	R2	A25	S13	A10
B8	D31	D2	D13	H17	VSS	N15	LOCK#	R3	VCC	S14	VSS
B9	VCC	D3	D17	J1	VCC	N16	M/IO#	R4	VSS	S15	A6
B10	NC	D15	A20M#	J2	D5	N17	W/R#	R5	A18	S16	A4
B11	VCC	D16	BS8#	J3	D16	P1	D0	R6	VCC	S17	ADS#

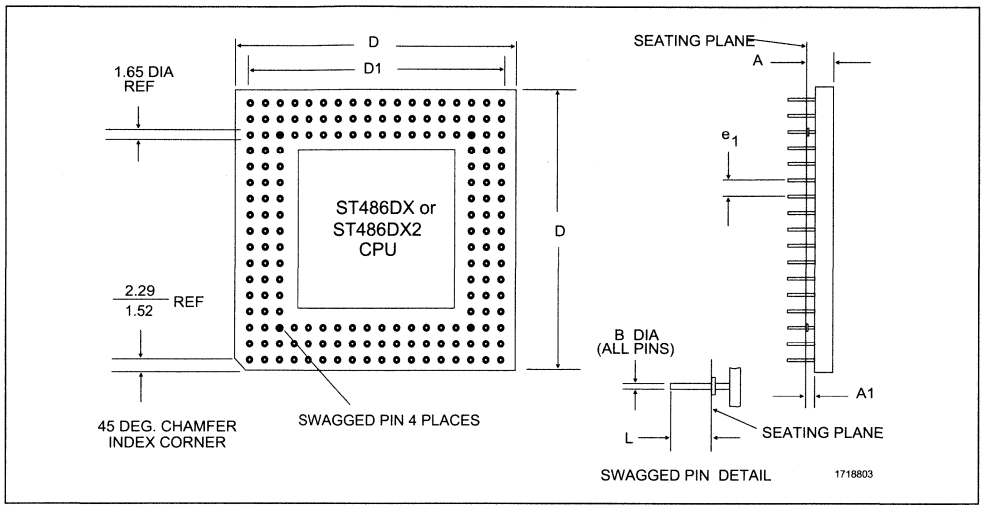


Figure 5 - 2. 168-Pin PGA Package

Table 5 - 4. 168-Pin PGA Package Dimensions

SYMBOL	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	3.65	4.57	0.140	0.180
A1	1.14	1.40	0.045	0.055
B	0.43	0.51	0.017	0.020
D	44.07	44.83	1.735	1.765
D1	40.51	40.77	1.595	1.605
e1	2.29	2.79	0.090	0.110
L	2.54	3.30	0.110	0.120

5.2 208-Lead QFP Package

Pin Assignments

The pin assignments for the ST486DX 208-lead QFP (Quad Flat Pack) package are shown in Figure 5-3. The signal names are shown in Table 5-5 sorted by signal name and in Table 5-6 sorted by pin numbers. Package dimensions are shown in Figure 5-4 and Table 5-7.

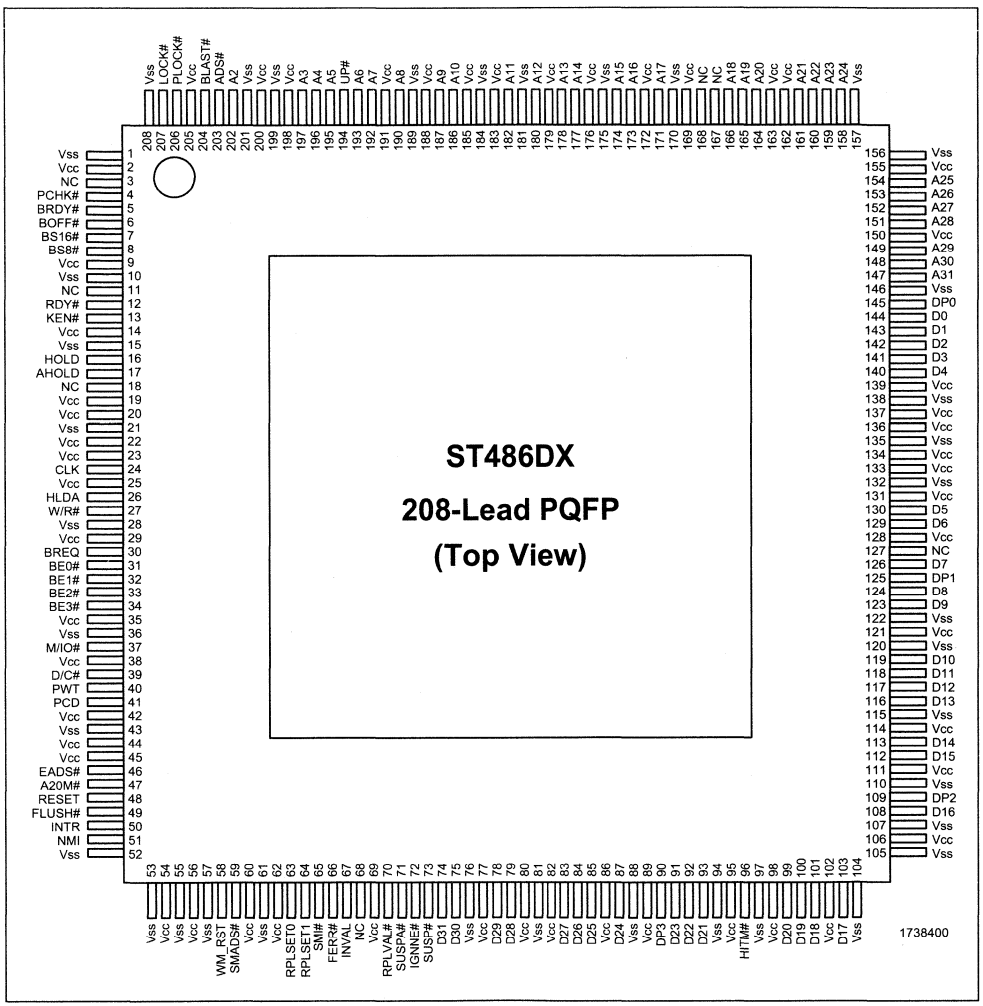


Figure 5 - 3. 208-Lead QFP Package Pin Assignments

Table 5 - 5. 208-Lead QFP Package Pins Sorted by Signal Name

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
A2	202	BE2#	33	D26	84	RDY#	12	Vcc	98	Vss	52
A3	197	BE3#	34	D27	83	RESET	48	Vcc	102	Vss	53
A4	196	BLAST#	204	D28	79	RPLSET0	63	Vcc	106	Vss	55
A5	195	BOFF#	6	D29	78	RPLSET1	64	Vcc	111	Vss	57
A6	193	BRDY#	5	D30	75	RPLVAL#	70	Vcc	114	Vss	61
A7	192	BREQ	30	D31	74	SMADS#	59	Vcc	121	Vss	76
A8	190	BS16#	7	D/C#	39	SMI#	65	Vcc	128	Vss	81
A9	187	BS8#	8	DP0	145	SUSP#	73	Vcc	131	Vss	88
A10	186	CLK	24	DP1	125	SUSPA#	71	Vcc	133	Vss	94
A11	182	D0	144	DP2	109	UP#	194	Vcc	134	Vss	97
A12	180	D1	143	DP3	90	Vcc	2	Vcc	136	Vss	104
A13	178	D2	142	EADS#	46	Vcc	9	Vcc	137	Vss	105
A14	177	D3	141	FERR#	66	Vcc	14	Vcc	139	Vss	107
A15	174	D4	140	FLUSH#	49	Vcc	19	Vcc	150	Vss	110
A16	173	D5	130	HITM#	96	Vcc	20	Vcc	155	Vss	115
A17	171	D6	129	HLDA	26	Vcc	22	Vcc	162	Vss	120
A18	166	D7	126	HOLD	16	Vcc	23	Vcc	163	Vss	122
A19	165	D8	124	IGNNE#	72	Vcc	25	Vcc	169	Vss	132
A20	164	D9	123	INTR	50	Vcc	29	Vcc	172	Vss	135
A20M#	47	D10	119	INVAL	67	Vcc	35	Vcc	176	Vss	138
A21	161	D11	118	KEN#	13	Vcc	38	Vcc	179	Vss	146
A22	160	D12	117	LOCK#	207	Vcc	42	Vcc	183	Vss	156
A23	159	D13	116	M/IO#	37	Vcc	44	Vcc	185	Vss	157
A24	158	D14	113	NC	3	Vcc	45	Vcc	188	Vss	170
A25	154	D15	112	NC	11	Vcc	54	Vcc	191	Vss	175
A26	153	D16	108	NC	18	Vcc	56	Vcc	198	Vss	181
A27	152	D17	103	NC	68	Vcc	60	Vcc	200	Vss	184
A28	151	D18	101	NC	127	Vcc	62	Vcc	205	Vss	189
A29	149	D19	100	NC	167	Vcc	69	Vss	1	Vss	199
A30	148	D20	99	NC	168	Vcc	77	Vss	10	Vss	201
A31	147	D21	93	NMI	51	Vcc	80	Vss	15	Vss	208
ADS#	203	D22	92	PCD	41	Vcc	82	Vss	21	WM_RST	58
AHOLD	17	D23	91	PCHK	4	Vcc	86	Vss	28	W/R#	27
BE0#	31	D24	87	PLOCK#	206	Vcc	89	Vss	36		
BE1#	32	D25	85	PWT	40	Vcc	95	Vss	43		

Table 5 - 6. 208-Lead QFP Package Signal Sorted by Pins Number

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	Vss	36	Vss	71	SUSPA#	106	Vcc	141	D3	176	Vcc
2	Vcc	37	M/IO#	72	IGNNE#	107	Vss	142	D2	177	A14
3	NC	38	Vcc	73	SUSP#	108	D16	143	D1	178	A13
4	PCHK#	39	D/C#	74	D31	109	DP2	144	D0	179	Vcc
5	BRDY#	40	PWT	75	D30	110	Vss	145	DP0	180	A12
6	BOFF#	41	PCD	76	Vss	111	Vcc	146	Vss	181	Vss
7	BS16#	42	Vcc	77	Vcc	112	D15	147	A31	182	A11
8	BS8#	43	Vss	78	D29	113	D14	148	A30	183	Vcc
9	Vcc	44	Vcc	79	D28	114	Vcc	149	A29	184	Vss
10	Vss	45	Vcc	80	Vcc	115	Vss	150	Vcc	185	Vcc
11	NC	46	EADS#	81	Vss	116	D13	151	A28	186	A10
12	RDY#	47	A20M#	82	Vcc	117	D12	152	A27	187	A9
13	KEN#	48	RESET	83	D27	118	D11	153	A26	188	Vcc
14	Vcc	49	FLUSH#	84	D26	119	D10	154	A25	189	Vss
15	Vss	50	INTR	85	D25	120	Vss	155	Vcc	190	A8
16	HOLD	51	NMI	86	Vcc	121	Vcc	156	Vss	191	Vcc
17	AHOLD	52	Vss	87	D24	122	Vss	157	Vss	192	A7
18	NC	53	Vss	88	Vss	123	D9	158	A24	193	A6
19	Vcc	54	Vcc	89	Vcc	124	D8	159	A23	194	UP#
20	Vcc	55	Vss	90	DP3	125	DP1	160	A22	195	A5
21	Vss	56	Vcc	91	D23	126	D7	161	A21	196	A4
22	Vcc	57	Vss	92	D22	127	NC	162	Vcc	197	A3
23	Vcc	58	WM_RST	93	D21	128	Vcc	163	Vcc	198	Vcc
24	CLK	59	SMADS#	94	Vss	129	D6	164	A20	199	Vss
25	Vcc	60	Vcc	95	Vcc	130	D5	165	A19	200	Vcc
26	HLDA	61	Vss	96	HITM#	131	Vcc	166	A18	201	Vss
27	W/R#	62	Vcc	97	Vss	132	Vss	167	NC	202	A2
28	Vss	63	RPLSET0	98	Vcc	133	Vcc	168	NC	203	ADS#
29	Vcc	64	RPLSET1	99	D20	134	Vcc	169	Vcc	204	BLAST#
30	BREQ	65	SMI#	100	D19	135	Vss	170	Vss	205	Vcc
31	BE0#	66	FERR#	101	D18	136	Vcc	171	A17	206	PLOCK#
32	BE1#	67	INVAL	102	Vcc	137	Vcc	172	Vcc	207	LOCK#
33	BE2#	68	NC	103	D17	138	Vss	173	A16	208	Vss
34	BE3#	69	Vcc	104	Vss	139	Vcc	174	A15		
35	Vcc	70	RPLVAL#	105	Vss	140	D4	175	Vss		

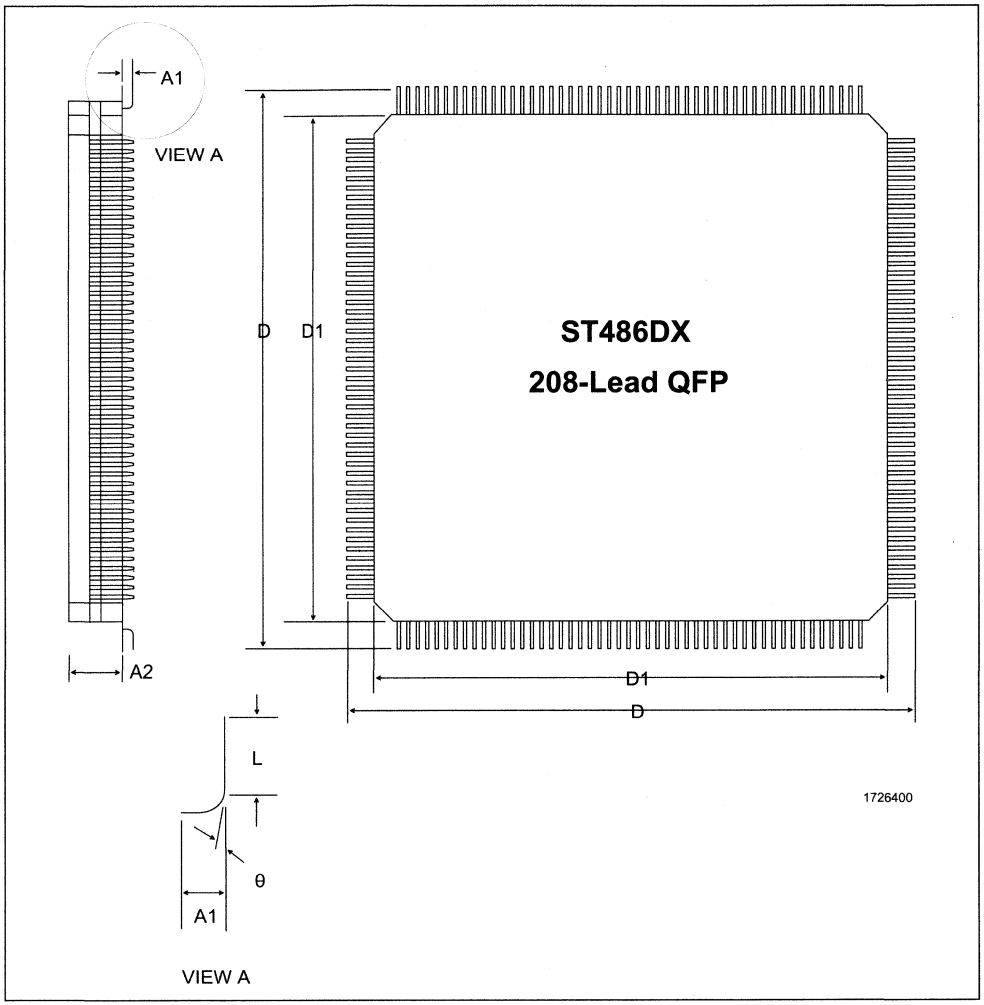


Figure 5 - 4. 208-Lead QFP Package

Table 5 - 7. 208-Lead QFP Package Dimensions

SYMBOL	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A1	0.13	0.33	0.005	0.013
A2	3.27	3.47	0.129	0.137
D	30.45	30.75	1.198	1.21
D1	27.9	28.1	1.098	1.106
L	0.4	0.6	0.015	0.023
θ	0°	7°		

5.3 Thermal Characteristics

The ST486DX/DX2 is designed to operate when case temperature is between 0° - 85°C. The case temperature is measured on the top center of the package. The maximum die temperature ($T_{J\,MAX}$) and the maximum ambient temperature (T_{aMAX}) can be calculated using the following equations.

$$T_{j\,MAX} = T_c + (P_{MAX} \times \theta_{jc})$$

$$T_{a\,MAX} = T_j - (P_{MAX} \times \theta_{ja})$$

where:

$T_{j\,MAX}$ = Maximum average junction temperature (°C)

T_c = Case temperature at top center of package (°C)

P_{MAX} = Maximum device power dissipation (W)

θ_{jc} = Junction-to-case thermal resistance (°C/W)

$T_{a\,MAX}$ = Maximum ambient temperature (°C)

T_j = Average junction temperature (°C)

θ_{ja} = Junction-to-ambient thermal resistance (°C/W)

PGA Package

Table 5-8 lists the junction-to-ambient and junction-to-case thermal resistances for the PGA package. Tables 5-9 and 5-10 lists the maximum ambient temperatures permitted for various clock frequencies and airflows for the PGA Package for Vcc equal to 3.6 and 5.25 volts respectively. Package dimensions for the heatsink used for the thermal analysis are shown in Figure 5-5 (Page 149) and Table 5-11 (Page 149).

Table 5 - 8. PGA Package Thermal Resistance and Airflow

AIRFLOW (FT/MIN)	PGA THERMAL RESISTANCE (C/W)			
	WITH HEATSINK		WITHOUT HEATSINK	
	θja	θjc	θja	θjc
0	12	3.5	17	3.0
200	8	3.5	15	3.0
400	6	3.5	12	3.0
600	4.5	3.5	10	3.0
800	4	3.5	9	3.0

Table 5 - 9. PGA Package Maximum Ambient Temperature (T_A) with V_{CC} = 3.6 V

CPU INTERNAL CLOCK FREQUENCY	HEATSINK	AIRFLOW				
		0 (FT/MIN)	200 (FT/MIN)	400 (FT/MIN)	600 (FT/MIN)	800 (FT/MIN)
33 MHz	No	52 °C	57 °C	64° C	69 °C	71° C
40 MHz	No	50 °C	55° C	63° C	68 °C	70° C
50 MHz	No	47 °C	52° C	60 °C	66 °C	69 °C
66 MHz	No	42 °C	48° C	57 °C	63° C	66 °C
66 MHz	Yes	59 °C	71 °C	77 °C	82° C	83 °C
80 MHz	No	37 °C	44 °C	54° C	61° C	64 °C
80 MHz	Yes	56 °C	69 °C	76 °C	81 °C	83° C
Note: Table refers to temperature specifications only and does not necessarily reflect part availability.						

Table 5 - 10. PGA Package Maximum Ambient Temperature (T_A) with V_{CC} =5.25 V

CPU INTERNAL CLOCK FREQUENCY	HEATSINK	AIRFLOW				
		0 (FT/MIN)	200 (FT/MIN)	400 (FT/MIN)	600 (FT/MIN)	800 (FT/MIN)
33 MHz	No	17 °C	26 °C	41° C	51 °C	55° C
33 MHz	Yes	43° C	63 °C	73° C	80 °C	82 °C
40 MHz	Yes	39° C	60 °C	71° C	79 °C	82 °C
50 MHz	Yes	32° C	57° C	69 °C	78 °C	81 °C
66 MHz	Yes	26 °C	53° C	67 °C	78 °C	81° C

Note: Table refers to temperature specifications only and does not necessarily reflect part availability.

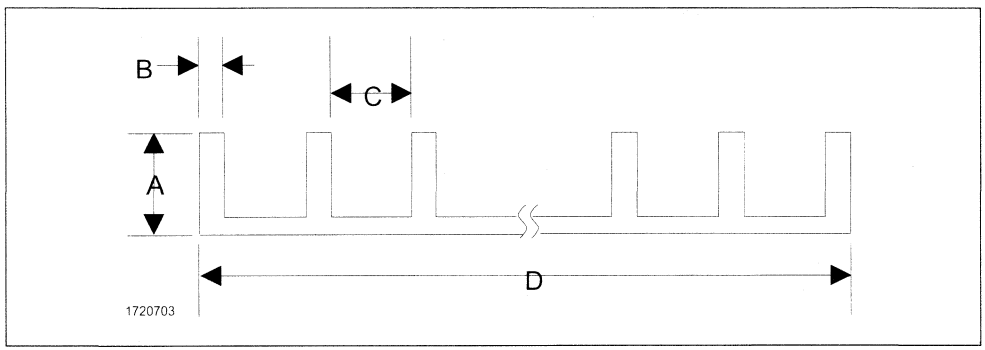


Figure 5 - 5. Heatsink for PGA Package

Table 5 - 11. Typical PGA Heatsink Dimensions

SYMBOL	MILLIMETERS	INCHES
A	6.1	0.24
B	1.3	0.05
C	4.8	0.19
D	39.1	1.54

QFP Package

Table 5-12 lists the junction-to-ambient and junction-to-case thermal resistances for the QFP package without a heat sink. Table 5-13 lists the maximum ambient temperatures permitted for various clock frequencies and airflows for the QFP Package for V_{CC} equal to 3.6 volts. These QFP package thermal characteristics assume that the package is soldered to a four-layer printed circuit board.

Table 5 - 12. QFP Package Thermal Resistance and Airflow without Heatsink

AIRFLOW (FT/MIN)	QFP THERMAL RESISTANCE (°C/W)	
	θ_{ja}	θ_{jc}
0	21	3.5
100	17	3.5

Table 5 - 13. QFP Package Maximum Ambient Temperature (T_A) with V_{CC} = 3.6 Volts and no Heatsink

CPU INTERNAL CLOCK FREQUENCY	AIRFLOW	
	0 (FT/MIN)	100 (FT/MIN)
33 MHz	44° C	54 °C
40 MHz	42 °C	52 °C
50 MHz	37 °C	48 °C
66 MHz	31° C	43° C
80 MHz	25° C	39° C
Note: Table refers to temperature specifications only and does not necessarily reflect part availability.		

INSTRUCTION SET

6.0 INSTRUCTION SET

This section summarizes the ST486DX/DX2 instruction set and provides detailed information on the instruction encodings. All instructions are listed in the CPU Instruction Set Summary Table (Table 6-17, Page 165), and the FPU Instruction Set Summary Table (Table 6-19, Page 181). These tables provide information on the instruction encoding, and the instruction clock counts for each instruction. The clock count values for both tables are based on the assumptions described in Section 6.3.

6.1 Instruction Set Summary

Depending on the instruction, the ST486DX/DX2 CPU instructions follow the general instruction format shown in Figure 6-1. These instructions vary in length and can start at any byte address. An instruction consists of one or more bytes that can include: prefix byte(s), at least one opcode byte(s), mod r/m byte, s-i-b byte, address displacement byte(s) and immediate data byte(s). An instruction can be as short as one byte and as long as 15 bytes. If there are more than 15 bytes in the instruction a general protection fault (error code of 0) is generated.

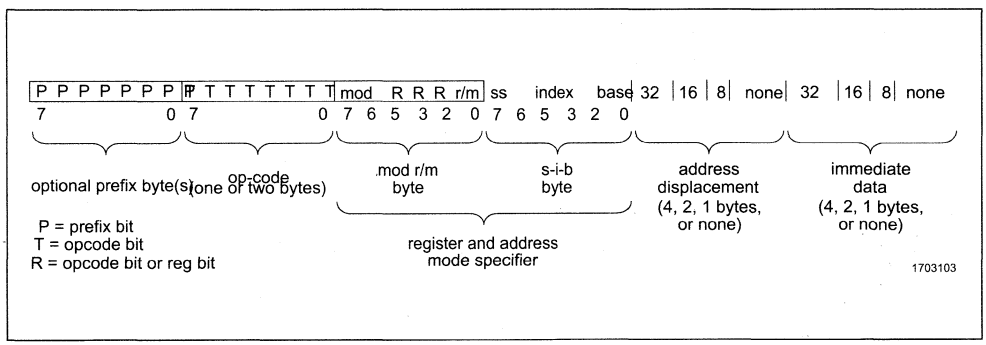


Figure 6 - 1. Instruction Set Format

6.2 General Instruction Fields

The fields in the general instruction format at the byte level are listed in Table 6-1.

Table 6 - 1. Instruction Fields

FIELD NAME	DESCRIPTION	WIDTH
Optional Prefix Byte(s)	Specifies segment register override, address and operand size, repeat elements in string instruction, LOCK# assertion.	1 or more bytes
Opcode Byte(s)	Identifies instruction operation.	1 or 2 bytes
mod and r/m Byte	Address mode specifier.	1 byte
s-i-b Byte	Scale factor, Index and Base fields.	1 byte
Address Displacement	Address displacement operand.	1, 2 or 4 bytes
Immediate data	Immediate data operand.	1, 2 or 4 bytes

6.2.1 Optional Prefix Bytes(s)

Prefix bytes can be placed in front of any instruction. The prefix modifies the operation of the next instruction only. When more than one prefix is used, the order is not important. There are five type of prefixes as follows:

1. Segment Override explicitly specifies which segment register an instruction will use for effective address calculation.
2. Address Size switches between 16- and 32-bit addressing. Selects the inverse of the default.
3. Operand Size switches between 16- and 32-bit operand size. Selects the inverse of the default.
4. Repeat is used with a string instruction which causes the instruction to be repeated for each element of the string.
5. Lock is used to assert the hardware LOCK# signal during execution of the instruction.

Table 6-2 lists the encodings for each of the available prefix bytes. The operand size and address size prefixes allow the individual overriding of the default value for operand size and effective address size. The presence of these prefixes select the opposite (non-default) operand size and/or effective address size as the case may be.

Table 6 - 2. Instruction Prefix Summary

PREFIX	ENCODING	DESCRIPTION
ES:	26h	Override segment default, use ES for memory operand
CS:	2Eh	Override segment default, use CS for memory operand
SS:	36h	Override segment default, use SS for memory operand
DS:	3Eh	Override segment default, use DS for memory operand
FS:	64h	Override segment default, use FS for memory operand
GS:	65h	Override segment default, use GS for memory operand
Operand Size	66h	Make operand size attribute the inverse of the default
Address Size	67h	Make address size attribute the inverse of the default
LOCK	F0h	Assert LOCK# hardware signal.
REPNE	F2h	Repeat the following string instruction.
REP/REPE	F3h	Repeat the following string instruction.

6.2.2 Opcode Byte(s)

The opcode field is either one or two bytes in length and may be further defined by additional bits in the mod r/m byte. The opcode field specifies the operation to be performed by the instruction. Some operations have more than one opcode, each specifying a different form of the operation. Some opcodes name instruction groups. For example, opcode 0x80 names a group of operations that have an immediate operand, and a register or memory operand. The opcodes are given in hex values unless shown within brackets ([ ]). Values within brackets are given in binary. The reg field may appear in the second opcode byte or in the mod r/m byte.

6.2.2.1 w Field

The 1-bit w field selects the operand size during 16 and 32 bit data operations.

Table 6 - 3. w Field Encoding

w FIELD	OPERAND SIZE 16-BIT DATA OPERATIONS	OPERAND SIZE 32-BIT DATA OPERATIONS
0	8 Bits	8 Bits
1	16 Bits	32 Bits

6.2.2.2 d Field

The d field determinds which operand is taken as the source operand and which operand is taken as the destination.

Table 6 - 4. d Field Encoding

d FIELD	DIRECTION OF OPERATON	SOURCE OPERAND	DESTINATION OPERAND
0	Register → Register or Register → Memory	reg	mod r/m or mod ss-index-base
1	Register → Register or Memory → Register	mod r/m or mod ss-index-base	reg

6.2.2.3 eee Field

The eee field is used to select the control, debug and test registers in the MOV instructions. The type of register and base registers selected by the eee field are listed in Table 6-5. The values shown in Table 6-5 are the only valid encodings for the eee bits.

Table 6 - 5. eee Field Encoding

eee FIELD	REGISTER TYPE	BASE REGISTER
000	Control Register	CR0
010	Control Register	CR2
011	Control Register	CR3
000	Debug Register	DR0
001	Debug Register	DR1
010	Debug Register	DR2
011	Debug Register	DR3
110	Debug Register	DR6
111	Debug Register	DR7
011	Test Register	TR3
100	Test Register	TR4
101	Test Register	TR5
110	Test Register	TR6
111	Test Register	TR7

6.2.3 mod and r/m Byte

The mod and r/m fields, within the mod r/m byte, select the type of memory addressing to be used. Some instructions use a fixed addressing mode (e.g., PUSH or POP) and therefore, these fields are not present. Table 6-6 lists the addressing method when 16-bit addressing is used and a mod r/m byte is present. Some mod r/m field encodings are dependent on the w field and are shown in Table 6-7 (Page 159).

Table 6 - 6. mod r/m Field Encoding

mod and r/m fields	16-BIT ADDRESS MODE with mod r/m Byte	32-BIT ADDRESS MODE with mod r/m Byte and No s-i-b Byte Present
00 000	DS:[BX+SI]	DS:[EAX]
00 001	DS:[BX+DI]	DS:[ECX]
00 010	DS:[BP+SI]	DS:[EDX]
00 011	DS:[BP+DI]	DS:[EBX]
00 100	DS:[SI]	s-i-b is present (See 6.2.7)
00 101	DS:[DI]	DS:[d32]
00 110	DS:[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	DS:[BP+SI+d8]	DS:[EDX+d8]
01 011	DS:[BP+DI+d8]	DS:[EBX+d8]
01 100	DS:[SI+d8]	s-i-b is present (See 6.2.7)
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	DS:[BP+SI+d16]	DS:[EDX+d32]
10 011	DS:[BP+DI+d16]	DS:[EBX+d32]
10 100	DS:[SI+d16]	s-i-b is present (See 6.2.7)
10 101	DS:[DI+d16]	SS:[EBP+d32]
10 110	SS:[BP+d16]	DS:[ESI+d32]
10 111	DS:[BX+d16]	DS:[EDI+d32]
11000-11111	See Table 6-7	See Table 6-7

Table 6 - 7. mod r/m Field Encoding Dependent on w Field

mod r/m	16-BIT OPERATION w = 0	16-BIT OPERATION w = 1	32-BIT OPERATION w = 0	32-BIT OPERATION w = 1
11 000	AL	AX	AL	EAX
11 001	CL	CX	CL	ECX
11 010	DL	DX	DL	EDX
11 011	BL	BX	BL	EBX
11 100	AH	SP	AH	ESP
11 101	CH	BP	CH	EBP
11 110	DH	SI	DH	ESI
11 111	BH	DI	BH	EDI

6.2.3.1 reg Field

The reg field determines which general registers are to be used. The selected register is dependent on whether a 16 or 32 bit operation is current and the status of the w bit.

Table 6 - 8. reg Field

reg	16-BIT OPERATION w Field Not Present	32-BIT OPERATION w Field Not Present	16-BIT OPERATION w=0	16-BIT OPERATION w=1	32-BIT OPERATION w=0	32-BIT OPERATION w=1
000	AX	EAX	AL	AX	AL	EAX
001	CX	ECX	CL	CX	CL	ECX
010	DX	EDX	DL	DX	DL	EDX
011	BX	EBX	BL	BX	BL	EBX
100	SP	ESP	AH	SP	AH	ESP
101	BP	EBP	CH	BP	CH	EBP
110	SI	ESI	DH	SI	DH	ESI
111	DI	EDI	BH	DI	BH	EDI

6.2.3.2 sreg3 Field

The sreg3 field (Table 6-9) is 3-bit field that is similar to the sreg2 field, but allows use of the FS and GS segment registers.

Table 6 - 9. sreg3 Field Encoding

sreg3 FIELD	SEGMENT REGISTER SELECTED
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	undefined
111	undefined

6.2.3.3 sreg2 Field

The sreg2 field (Table 6-10) is a 2-bit field that allows one of the four 286-type segment registers to be specified.

Table 6 - 10. sreg2 Field Encoding

sreg2 FIELD	SEGMENT REGISTER SELECTED
00	ES
01	CS
10	SS
11	DS

6.2.4 s-i-b Byte

The s-i-b fields provide scale factor, indexing and a base field for address selection.

6.2.4.1 ss Field

The ss field (Table 6-11) specifies the scale factor used in the offset mechanism for address calculation. The scale factor multiplies the index value to provide one of the components used to calculate the offset address.

Table 6 - 11. ss Field Encoding

ss FIELD	SCALE FACTOR
00	x1
01	x2
01	x4
11	x8

6.2.4.2 index Field

The index field (Table 6-12) specifies the index register used by the offset mechanism for offset address calculation. When no register is used (index field = 100), the ss value must be 00 or the effective address is undefined.

Table 6 - 12. index Field Encoding

Index FIELD	INDEX REGISTER
000	EAX
001	ECX
010	EDX
011	EBX
100	none
101	EBP
110	ESI
111	EDI

6.2.4.3 Base Field

In Table 6-6, the note "s-i-b present" for certain entries forces the use of the mod and base field as listed in Table 6-13. The first two digits in the first column of Table 6-13 identifies the mod bits in the mod r/m byte. The last three digits in the first column of this table identifies the base fields in the s-i-b byte.

Table 6 - 13. mod base Field Encoding

mod FIELD WITHIN mode/rm BYTE	base FIELD WITHIN s-i-b BYTE	32-BIT ADDRESS MODE with mod r/m and s-i-b Bytes Present
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]

6.3 Instruction Set Tables

The ST486DX/DX2 instruction set is presented in two tables, the CPU Instruction Set (Table 6-17 on page 165) and the FPU Clock Count (Table 6-18 on page 181). Additional information concerning the FPU Clock Count Table is presented on page 180.

6.3.1 Assumptions Made in Determining Instruction Clock Count

The following assumptions have been made in presenting the clock count values for the individual instructions:

1. All clock counts refer to the internal CPU internal clock frequency. For example, the clock counts for a clock-doubled ST486DX2-50 refer to 50 MHz clocks while the external clock is 25MHz.
2. The instruction has been prefetched, decoded and is ready for execution.
3. Bus cycles do not require wait states.
4. There are no local bus HOLD requests delaying processor access to the bus.
5. No exceptions are detected during instruction execution.
6. If an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock count shown. However, if the effective address calculation uses two general register components, add one clock to the clock count shown.
7. All clock counts assume aligned 32-bit memory/IO operands.
8. If instructions access a 32-bit operand on odd addresses, add one clock for read or write and add two clocks for read and write.
9. For non-cached memory accesses, add two clocks (ST486DX) or four clocks (ST486DX2), assuming zero wait state memory accesses.
10. Locked cycles are not cacheable. Therefore, using the LOCK prefix with an instruction adds additional clocks as specified in paragraph 9 above.

6.3.2 CPU Instruction Set Summary Table Abbreviations

The clock counts listed in the CPU Instruction Set Summary Table are grouped by operating mode and whether there is a register/cache hit or a cache miss. In some cases, more than one clock count is shown in a column for a given instruction, or a variable is used in the clock count. The abbreviations used for these conditions are listed in Table 6-14.

Table 6 - 14. CPU Clock Count Abbreviations

CLOCK COUNT SYMBOL	EXPLANATION
/	Register operand/memory operand.
n	Number of times operation is repeated.
L	Level of the stack frame.
I	Conditional jump taken I Conditional jump not taken (e.g. "4I1" = 4 clocks if jump taken, 1 clock if jump not taken)
\	CPL ≤ IOPL \ CPL IOPL (where CPL = Current Privilege Level, IOPL = I/O Privilege Level)

6.3.3 CPU Instruction Set Summary Table Flags Table

The CPU Instruction Set Summary Table lists nine flags that are affected by the execution of instructions. The conventions shown in Table 6-15 are used to identify the different flags. Table 6-16 lists the conventions used to indicate what action the instruction has on the particular

Table 6 - 15. Flag Abbreviations

ABBREVIATION	NAME OF FLAG
OF	Overflow Flag
DF	Direction Flag
IF	Interrupt Enable Flag
TF	Trap Flag
SF	Sign Flag
ZF	Zero Flag
AF	Auxiliary Flag
PF	Parity Flag
CF	Carry Flag

Table 6 - 16. Action of Instruction on Flag

INSTRUCTION TABLE SYMBOL	ACTION
x	Flag is modified by the instruction.
-	Flag is not changed by the instruction.
0	Flag is reset to "0".
1	Flag is set to "1".

Table 6 - 17. Instruction Set Summary

INSTRUCTION	OPCODE	FLAGS									REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg/Cache Hit	Reg/Cache Hit	Real Mode	Protected Mode
AAA *ASCII Adjust AL after Add*	37	-	-	-	-	-	-	x	-	x	4	4		
AAD *ASCII Adjust AX before Divide*	D5 0A	-	-	-	-	x	x	-	x	-	4	4		
AAM *ASCII Adjust AX after Multiply*	D4 0A	-	-	-	-	x	x	-	x	-	16	16		
AAS *ASCII Adjust AL after Subtract*	3F	-	-	-	-	-	-	x	-	x	4	4		
ADC *Add with Carry*		x	-	-	-	x	x	x	x	x			b	h
Register to Register	1 [00dw] [11 reg r/m]										1	1		
Register to Memory	1 [000w] [mod reg r/m]										3	3		
Memory to Register	1 [001w] [mod reg r/m]										3	3		
Immediate to Register/Memory	8 [00sw] [mod 010 r/m]#										1/3	1/3		
Immediate to Accumulator	1 [010w] #										1	1		
ADD *Integer Add*		x	-	-	-	x	x	x	x	x			b	h
Register to Register	0 [00dw] [11 reg r/m]										1	1		
Register to Memory	0 [000w] [mod reg r/m]										3	3		
Memory to Register	0 [001w] [mod reg r/m]										3	3		
Immediate to Register/Memory	8 [00sw] [mod 000 r/m]#										1/3	1/3		
Immediate to Accumulator	0 [010w] #										1	1		
AND *Boolean AND*		0	-	-	-	x	x	-	x	0			b	h
Register to Register	2 [00dw] [11 reg r/m]										1	1		
Register to Memory	2 [000w] [mod reg r/m]										3	3		
Memory to Register	2 [001w] [mod reg r/m]										3	3		
Immediate to Register/Memory	8 [00sw] [mod 100 r/m]#										1/3	1/3		
Immediate to Accumulator	2 [010w] #										1	1		
ARPL *Adjust Requested Privilege Level*		-	-	-	-	-	x	-	-	-			a	h
From Register/Memory	63 [mod reg r/m]											6/10		
BOUND *Check Array Boundaries*	62 [mod reg r/m]	-	-	-	-	-	-	-	-	-			b,e	g,h,k,j,k,r
If Out of Range (Int 5)											11+int	11+int		
If In Range											11	11		
BSF *Scan Bit Forward*		-	-	-	-	-	x	-	-	-			b	h
Register/Memory, Register	0F BC [mod reg r/m]										5/7+n	5/7+n		
BSR *Scan Bit Reverse*		-	-	-	-	-	x	-	-	-			b	h
Register/Memory, Register	0F BC [mod reg r/m]										5/7+n	5/7+n		
BSWAP Byte Swap	0F C[1 reg]	-	-	-	-	-	-	-	-	-	4	4		
BT *Test Bit*		-	-	-	-	-	-	-	-	x			b	h
Register/Memory, Immediate	0F BA [mod 100 r/m]#										3/4	3/4		
Register/Memory, Register	0F A3 [mod reg r/m]										3/6	3/6		
BTC *Test Bit and Complement*		-	-	-	-	-	-	-	-	x			b	h
Register/Memory, Immediate	0F BA [mod 111 r/m]#										4/5	4/5		
Register/Memory, Register	0F BB [mod reg r/m]										5/8	5/8		

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full) - = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/Cache Hit	Reg/Cache Hit	Real Mode	Protected Mode
BTR *Test Bit and Reset* Register/Memory, Immediate Register/Memory, Register	0F BA [mod 110 r/m]# 0F B3 [mod reg r/m]	- - - - - - - - - x	 4/5 5/8	 5/8	b	h
BTS *Test Bit and Set* Register/Memory Register (short form)	0F BA [mod 101 r/m] 0F AB [mod reg r/m]	- - - - - - - - - x	 3/5 4/7	 3/5 4/7	b	h
CALL *Subroutine Call* Direct Within Segment Register/Memory Indirect Within Segment Direct Intersegment Call Gate to Same Privilege Call Gate to Different Privilege No P Call Gate to Different Privilege x P's 16-bit Task to 16-bit TSS 16-bit Task to 32-bit TSS 16-bit Task to V86 Task 32-bit Task to 16-bit TSS 32-bit Task to 32-bit TSS 32-bit Task to V86 Task Indirect Intersegment Call Gate to Same Privilege Call Gate to Different Privilege No P Call Gate to Different Privilege x P's 16-bit Task to 16-bit TSS 16-bit Task to 32-bit TSS 16-bit Task to V86 Task 32-bit Task to 16-bit TSS 32-bit Task to 32-bit TSS 32-bit Task to V86 Task	 E8 +++ FF [mod 010 r/m] 9A [unsigned full offset, selector] FF [mod 011 r/m] 	- - - - - - - - - -	 7 8/9 12 14 	 7 8/9 30 41 83 81+4x 235 262 179 238 265 182 14 43 85 86+4x 237 264 181 240 267 184	b	h,j,k,r
CBW *Convert Byte to Word*	98	- - - - - - - - - -	3	3		
CDQ *Convert Doubleword to Quadword*	99	- - - - - - - - - -	1	1		
CLC *Clear Carry Flag*	F8	- - - - - - - - - 0	1	1		
CLD *Clear Direction Flag*	FC	- 0 - - - - - - - -	1	1		
CLI *Clear Interrupt Flag*	FA	- - 0 - - - - - - -	7	7		m
CLTS *Clear Task Switched Flag*	0F 06	- - - - - - - - - -	5	5	c	l
CMC *Complement the Carry Flag*	F5	- - - - - - - - - x	1	1		
P = Parameters	# = immediate data ++ = 16-bit displacement x = modified + = 8-bit displacement +++ = 32 bit displacement (full)- = unchanged					

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
CMP *Compare Integers* Register to Register Register to Memory Memory to Register Immediate to Register/Memory Immediate to Accumulator	 3 [10dw] [11 reg r/m] 3 [101w] [mod reg r/m] 3 [100w] [mod reg r/m] 8 [00sw] [mod 111 r/m] # 3 [110w] #	x - - - x x x x x	 1 3 3 1/3 1	 1 3 3 1/3 1	b	h
CMPS *Compare String*	A [011w]	x - - - x x x x x	7	7	b	h
CMPXCHG *Compare and Exchange* Register1, Register2 Memory, Register	 0F B [000w] [11 reg2 reg1] 0F B [000w] [mod reg r/m]	x - - - x x x x x	 5 7	 5 7		
CWD *Convert Word to Doubleword*	99	- - - - - - - - -	1	1		
CWDE *Convert Word to Doubleword*	98	- - - - - - - - -	3	3		
DAA *Decimal Adjust AL after Add*	27	- - - - x x x x x	4	4		
DAS *Decimal Adjust AL after Subtract*	2F	- - - - x x x x x	4	4		
DEC *Decrement by 1* Register/Memory Register (short form)	 F [111w] [mod 001 r/m] 4 [1 reg]	x - - - x x x x -	 1/3 1	 1/3 1	b	h
DIV *Unsigned Divide* Accumulator by Register/Memory Divisor: Byte Word Doubleword	F [011w] [mod 110 r/m]	- - - - - - - - -	 14/15 22/23 38/39	 14/15 22/23 38/39	b,e	e,h
ENTER *Enter New Stack Frame* Level = 0 Level = 1 Level (L) 1	C8 ++[8-bit Level]	- - - - - - - - -	 7 10 6+4*L	 7 10 6+4*L	b	h
HLT *Halt*	F4	- - - - - - - - -	3	3		1
IDIV *Integer (Signed) Divide* Accumulator by Register/Memory Divisor: Byte Word Doubleword	F [011w] [mod 111 r/m]	- - - - - - - - -	 19/20 27/28 43/44	 19/20 27/28 43/44	b,e	e,h

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full) - = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
IMUL *Integer (Signed) Multiply*		x - - - - - - - x			b	h
Accumulator by Register/Memory	F [011w] [mod 101 r/m]					
Multiplier: Byte			3/5	3/5		
Word			3/5	3/5		
Doubleword			7/9	7/9		
Register with Register/Memory	0F AF [mod reg r/m]					
Multiplier: Byte			3/5	3/5		
Word			3/5	3/5		
Doubleword			7/9	7/9		
Register/Memory with Immediate to Register2	6 [10s1] [mod reg r/m] #					
Multiplier: Byte			3/5	3/5		
Word			3/5	3/5		
Doubleword			7/9	7/9		
IN *Input from I/O Port*		- - - - - - - - - -				m
Fixed Port	E [010w] [port number]		6/19	16		
Variable Port	E [110w]		6/19	16		
INC *Increment by 1*		x - - - x x x x -			b	h
Register/Memory	F [111w] [mod 000 r/m]		1/3	1/3		
Register (short form)	4 [0 reg]		1	1		
INS *Input String from I/O Port*	6 [110w]	- - - - - - - - - -	20	6/19	b	h,m
INT *Software Interrupt*		- x 0 - - - - - -			b,e	g,j,k,r
INT i	CD [i]		14			
Protected Mode:						
Interrupt or Trap to Same Privilege				49		
Interrupt or Trap to Different Privilege				77		
16-bit Task to 16-bit TSS by Task Gate				233		
16-bit Task to 32-bit TSS by Task Gate				260		
16-bit Task to V86 by Task Gate				177		
16-bit Task to 16-bit TSS by Task Gate				236		
32-bit Task to 32-bit TSS by Task Gate				263		
32-bit Task to V86 by Task Gate				180		
V86 to 16-bit TSS by Task Gate				236		
V86 to 32-bit TSS by Task Gate				263		
V86 to Privilege 0 by Trap Gate/Int Gate				93		
Continued on the next page...						

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full) - = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
INT *Software Interrupt* **(Continued)**		- X 0 - - - - - -			b,e	g,j,k,r
INT 3	CC		14			
Protected Mode:						
Interrupt or Trap to Same Privilege				49		
Interrupt or Trap to Different Privilege				77		
16-bit Task to 16-bit TSS by Task Gate16-bit				233		
Task to 32-bit TSS by Task Gate				260		
16-bit Task to V86 by Task Gate				177		
32-bit Task to 16-bit TSS by Task Gate				236		
32-bit Task to 32-bit TSS by Task Gate				180		
32-bit Task to V86 by Task Gate				236		
V86 to 16-bit TSS by Task Gate				263		
V86 to 32-bit TSS by Task Gate				93		
V86 to Privilege 0 by Trap Gate/Int Gate						
INTO	CE	- X 0 - - - - - -		1		
If OF==0			1			
If OF==1 (INT 4)			15			
Protected Mode:				49		
Interrupt or Trap to Same Privilege				77		
Interrupt or Trap to Different Privilege				233		
16-bit Task to 16-bit TSS by Task Gate				260		
16-bit Task to 32-bit TSS by Task Gate				177		
16-bit Task to V86 by Task Gate				236		
32-bit Task to 16-bit TSS by Task Gate				263		
32-bit Task to 32-bit TSS by Task Gate				180		
32-bit Task to V86 by Task Gate				236		
V86 to 16-bit TSS by Task Gate				263		
V86 to 32-bit TSS by Task Gate				93		
V86to Privilege 0 by Trap Gate/Int Gate						
INVD *Invalidate Cache*	0F 08	- - - - - - - - -	4	4		
INVLPG *Invalidate TLB Entry*	0F 01 [mod 111 r/m]	- - - - - - - - -	4	4		
RET *Interrupt Return*	CF	x x x x x x x x x				g,h,j,k,r
Real Mode			14			
Protected Mode:						
Within Task to Same Privilege				31		
Within Task to Different Privilege				66		
16-bit Task to 16-bit Task				229		
16-bit Task to 32-bit TSS				256		
16-bit Task to V86 Task				173		
32-bit Task to 16-bit TSS				232		
32-bit Task to 32-bit TSS				259		
32-bit Task to V86 Task				176		

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full)- = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF C	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
JB/JNAE/JC *Jump on Below/Not Above or Equal/Carry*		- - - - - - - - -	\			r
8-bit Displacement	72 +		4 | 1	6 | 1		
Full Displacement	0F 82 +++		4 | 1	6 | 1		
JBE/JNA *Jump on Below or Equal/Not Above*		- - - - - - - - -				r
8-bit Displacement	76 +		4 | 1	6 | 1		
Full Displacement	0F 86 +++		4 | 1	6 | 1		
JCXZ *Jump on CX Zero*	E3 +	- - - - - - - - -	7 | 3			r
JE/JZ *Jump on Equal/Zero*		- - - - - - - - -				r
8-bit Displacement	74 +		4 | 1	6 | 1		
Full Displacement	0F 84 +++		4 | 1	6 | 1		
JECXZ *Jump on ECX Zero*	E3 +	- - - - - - - - -	7 | 3			r
JL/JNGE *Jump on Less/Not Greater or Equal*		- - - - - - - - -				r
8-bit Displacement	7C +		4 | 1	6 | 1		
Full Displacement	0F 8C +++		4 | 1	6 | 1		
JLE/JNG *Jump on Less or Equal/Not Greater*		- - - - - - - - -				r
8-bit Displacement	7E +		4 | 1	6 | 1		
Full Displacement	0F 8E +++		4 | 1	6 | 1		
JMP *Unconditional Jump*		- - - - - - - - -			b	h,j,k,r
Short	EB +		4	6		
Direct within Segment	E9 +++		4	6		
Register/Memory Indirect Within Segment	FF [mod 100 r/m]		6/8	6/8		
Direct Intersegment	EA [full offset, selector]		9	26		
Call Gate Same Privilege Level				37		
16-bit Task to 16-bit TSS				238		
16-bit Task to 32-bit TSS				265		
16-bit Task to V86 Task				182		
32-bit Task to 16-bit TSS				241		
32-bit Task to 32-bit TSS				268		
32-bit Task to V86 Task				185		
Indirect Intersegment	FF [mod 101 r/m]		11	30		
Call Gate Same Privilege Level				39		
16-bit Task to 16-bit TSS				240		
16-bit Task to 32-bit TSS				267		
16-bit Task to V86 Task				184		
32-bit Task to 16-bit TSS				243		
32-bit Task to 32-bit TSS				270		
32-bit Task to V86 Task				187		

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full) - = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF C	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
JNB/JAE/JNC *Jump on Not Below/Above or Equal/Not Carry*		- - - - - - - - -		6 | 1		r
8-bit Displacement	73 +		4 | 1	6 | 1		
Full Displacement	0F 83 +++		4 | 1			
JNBE/JA *Jump on Not Below or Equal/Above*		- - - - - - - - -				r
8-bit Displacement	77 +		4 | 1	6 | 1		
Full Displacement	0F 87 +++		4 | 1	6 | 1		
JNE/JNZ *Jump on Not Equal/Not Zero*		- - - - - - - - -				r
8-bit Displacement	75 +		4 | 1	6 | 1		
Full Displacement	0F 85 +++		4 | 1	6 | 1		
JNL/JGE *Jump on Not Less/Greater or Equal*		- - - - - - - - -				r
8-bit Displacement	7D +		4 | 1	6 | 1		
Full Displacement	0F 8D +++		4 | 1	6 | 1		
JNLE/JG *Jump on Not Less or Equal/Greater*		- - - - - - - - -				r
8-bit Displacement	7F +		4 | 1	6 | 1		
Full Displacement	0F 8F +++		4 | 1	6 | 1		
JNO *Jump on Not Overflow*		- - - - - - - - -				r
8-bit Displacement	71 +		4 | 1	6 | 1		
Full Displacement	0F 81 +++		4 | 1	6 | 1		
JNP/JPO *Jump on Not Parity/Parity Odd*		- - - - - - - - -				r
8-bit Displacement	7B +		4 | 1	6 | 1		
Full Displacement	0F 8B +++		4 | 1	6 | 1		
JNS *Jump on Not Sign*		- - - - - - - - -				r
8-bit Displacement	79 +		4 | 1	6 | 1		
Full Displacement	0F 89 +++		4 | 1	6 | 1		
JO *Jump on Overflow*		- - - - - - - - -				r
8-bit Displacement	70 +		4 | 1	6 | 1		
Full Displacement	0F 80 +++		4 | 1	6 | 1		
JP/JPE *Jump on Parity/Parity Even*		- - - - - - - - -				r
8-bit Displacement	7A +		4 | 1	6 | 1		
Full Displacement	0F 8A +++		4 | 1	6 | 1		
JS *Jump on Sign*		- - - - - - - - -				r
8-bit Displacement	78 +		4 | 1	6 | 1		
Full Displacement	0F 88 +++		4 | 1	6 | 1		
LAHF *Load AH with Flags*	9F	- - - - - - - - -	2	2		
LAR *Load Access Rights*		- - - - - x - - -			a	g,h,j,p
From Register/Memory	0F 02 [mod reg r/m]			11/12		
LDS *Load Pointer to DS*	C5 [mod reg r/m]	- - - - - - - - -	6	19	b	h,i,j

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full) - = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF C	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
LEA *Load Effective Address* No Index Register With Index Register	8D [mod reg r/m]	- - - - - - - - -	 2 3	 2 3		
LEAVE *Leave Current Stack Frame*	C9	- - - - - - - - -	3	3	b	h
LES *Load Pointer to ES*	C4 [mod reg r/m]	- - - - - - - - -	6	19	b	h,i,j
LFS *Load Pointer to FS*	0F B4 [mod reg r/m]	- - - - - - - - -	6	19	b	h,i,j
LGDT *Load GDT Register*	0F 01 [mod 010 r/m]	- - - - - - - - -	9	9	b,c	h,l
LGS *Load Pointer to GS*	0F B5 [mod reg r/m]	- - - - - - - - -	6	19	b	h,i,j
LIDT *Load IDT Register*	0F 01 [mod 011 r/m]	- - - - - - - - -	9	9	b,c	h,l
LLDT *Load LDT Register* From Register/Memory	0F 00 [mod 010 r/m]	- - - - - - - - -		16/17	a	g,h,j,l
LMSW *Load Machine Status Word* From Register/Memory	0F 01 [mod 110 r/m]	- - - - - - - - -	5	5	b,c	h,l
LODS *Load String*	A [110 w]	- - - - - - - - -	4	4	b	h
LOOP *Offset Loop/No Loop*	E2 +	- - - - - - - - -	7|3	9|3		r
LOOPNZ/LOOPNE *Offset*	E0 +	- - - - - - - - -	7|3	9|3		r
LOOPZ/LOOPE *Offset*	E1 +	- - - - - - - - -	7|3	9|3		r
LSL *Load Segment Limit* From Register/Memory	0F 03 [mod reg r/m]	- - - - - x - - -		14/15	a	h,i,j,p
LSS *Load Pointer to SS*	0F B2 [mod reg r/m]	- - - - - - - - -	6	19	a	h,i,j
LTR *Load Task Register* From Register/Memory	0F 00 [mod reg r/m]	- - - - - - - - -		16/17	a	g,h,j,l
MOV *Move Data* Register to Register/Memory Register/Memory to Register Immediate to Register/Memory Immediate to Register (short form) Memory to Accumulator (short form) Accumulator to Memory (short form) Register/Memory to Segment Register Segment Register to Register/Memory	 8 [100w] [mod reg r/m] 8 [101w] [mod reg r/m] C [011w] [mod 000 r/m] # B [w reg] # A [000w] +++ A [001w] +++ 8E [mod sreg3 r/m] 8C [mod reg r/m]	- - - - - - - - -	 1/2 1/2 1/2 1 2 1/2 2/3 1/2	 1/2 1/2 1/2 1 2 1/2 15/16 1/2	b	h,i,j

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full) - = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF C	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
MOV *Move to/from Control/Debug/Test Regs*	0F 22 [11 eee reg]	- - - - - - - - -	11/3/3	11/3/3		l
Register to CR0/CR2/CR3	0F 20 [11 eee reg]		1/3/3	1/3/3		
CR0/CR2/CR3 to Register	0F 23 [11 eee reg]		1	1		
Register to DR0-DR3	0F 21 [11 eee reg]		3	3		
DR0-DR3 to Register	0F 23 [11 eee reg]		1	1		
Register to DR6-DR7	0F 21 [11 eee reg]		3	3		
DR6-DR7 to Register	0F 26 [11 eee reg]		5	5		
Register to TR3-5	0F 24 [11 eee reg]		5	5		
TR3-5 to Register	0F 26 [11 eee reg]		1	1		
Register to TR6-TR7	0F 24 [11 eee reg]		3	3		
TR6-TR7 to Register						
MOVS *Move String*	A [010w]	- - - - - - - - -	5	5	b	h
MOVSX *Move with Sign Extension*		- - - - - - - - -			b	h
Register from Register/Memory	0F B[111w] [mod reg r/m]		1/3	1/3		
MOVZX *Move with Zero Extension*		- - - - - - - - -			b	h
Register from Register/Memory	0F B[011w] [mod reg r/m]		2/3	2/3		
MUL *Unsigned Multiply*	F [011w] [mod 100 r/m]	x - - - - - - - x			b	h
Accumulator with Register/Memory						
Multiplier - Byte			3/5	3/5		
- Word			3/5	3/5		
- Doubleword			7/9	7/9		
NEG *Negate Integer*	F [011w] [mod 011 r/m]	x - - - x x x x x	1/3	1/3	b	h
NOP *No Operation*	90	- - - - - - - - -	1	1		
NOT *Boolean Complement*	F [011w] [mod 010 r/m]	- - - - - - - - -	1/3	1/3	b	h
OR *Boolean OR*		0 - - - x x x x 0			b	h
Register to Register	0 [10dw] [11 reg r/m]		1	1		
Register to Memory	0 [100w] [mod reg r/m]		3	3		
Memory to Register	0 [101w] [mod reg r/m]		3	3		
Immediate to Register/Memory	8 [000w] [mod 001 r/m] #		1/3	1/3		
Immediate to Accumulator	0 [110w] #		1	1		
OUT *Output to Port*		- - - - - - - - -				m
Fixed Port	E [011w] [port number]		18	4/17		
Variable Port	E [111w]		18	4/17		
OUTS *Output String*	6 [111w]	- - - - - - - - -	20	6/19	b	n,m
POP *Pop Value off Stack*		- - - - - - - - -				h,i,j
Register/Memory	8F [mod 000 r/m]		3/5	3/5		
Register (short form)	5 [1 reg]		3	3		
Segment Register (ES, CS, SS, DS)	[000 sreg2 110]		4	18		
Segment Register (ES, CS, SS, DS, FS, GS)	0F [10 sreg3 001]		4	18		

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full) - = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS									REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF	DF	IF	TF	SF	ZF	AF	PF	C	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
POPA *Pop All General Registers*	61	-	-	-	-	-	-	-	-	-	18	18	b	h
POPF *Pop Stack into FLAGS*	9D	x	x	x	x	x	x	x	x	x	4	4	b	h,n
PREFIX BYTES		-	-	-	-	-	-	-	-	-				m
Assert Hardware LOCK Prefix	F0													
Address Size Prefix	67													
Operand Size Prefix	66													
Segment Override Prefix														
CS	2E													
DS	3E													
ES	26													
FS	64													
GS	65													
SS	36													
PUSH *Push Value onto Stack*		-	-	-	-	-	-	-	-	-			b	h
Register/Memory	FF [mod 110 r/m]										2/4	2/4		
Register (short form)	5 [0 reg]										2	2		
Segment Register (ES, CS, SS, DS)	[000 sreg2 110]										2	2		
Segment Register (ES, CS, SS, DS, FS, GS)	0F [10 sreg3 000]										2	2		
Immediate	6 [10s0] #										2	2		
PUSHA *Push All General Registers*	60	-	-	-	-	-	-	-	-	-	17	17	b	h
PUSHF *Push FLAGS Register*	9C	-	-	-	-	-	-	-	-	-	2	2	b	h
RCL *Rotate Through Carry Left*		x	-	-	-	-	-	-	-	x			b	h
Register/Memory by 1	D [000w] [mod 010 r/m]										9/9	9/9		
Register/Memory by CL	D [001w] [mod 010 r/m]										9/9	9/9		
Register/Memory by Immediate	C [000w] [mod 010 r/m] #										9/9	9/9		
RCR *Rotate Through Carry Right*		x	-	-	-	-	-	-	-	x			b	h
Register/Memory by 1	D [000w] [mod 011 r/m]										9/9	9/9		
Register/Memory by CL	D [001w] [mod 011 r/m]										9/9	9/9		
Register/Memory by Immediate	C [000w] [mod 011 r/m] #										9/9	9/9		
REP INS *Input String*	F2 6[110w]	-	-	-	-	-	-	-	-	-	20 + 9n	5 + 9n\ 18 + 9n	b	h,m
REP LODS *Load String*	F2 A[110w]	-	-	-	-	-	-	-	-	-	4 + 5n	4 + 5n	b	h
REP MOVS *Move String*	F2 A[010w]	-	-	-	-	-	-	-	-	-	5 + 4n	5 + 4n	b	h
REP OUTS *Output String*	F2 6[111w]	-	-	-	-	-	-	-	-	-	20 + 4n	5 + 4n\ 18 + 4n	b	h,m
REP STOS *Store String*	F2 A[101w]	x	-	-	-	x	x	x	x	x	3 + 4n	3 + 4n	b	h
REPE CMPS *Compare String (Find non-match)*	F3 A[011w]										5 + 8n	5 + 8n	b	h
REPE SCAS *Scan String (Find non-AL/AX/EAX)*	F3 A[111w]	x	-	-	-	x	x	x	x	x	4 + 5n	4 + 5n	b	h

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full) - = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS									REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF	DF	IF	TF	SF	ZF	AF	PF	C	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
REPNE CMPS *Compare String (Find match)*	F2 A[011w]	x	-	-	-	x	x	x	x	x	5+8n	5+8n	b	h
REPNE SCAS *Scan String (Find AL/AX/EAX)*	F2 A[111w]	x	-	-	-	x	x	x	x	x	4+5n	4+5n	b	h
RET *Return from Subroutine*		-	-	-	-	-	-	-	-	-			b	g,h,j,k,r
Within Segment	C3										10	10		
Within Segment Adding Immediate to SP	C2 ++										10	10		
Intersegment	CB										13	26		
Intersegment Adding Immediate to SP	CA ++										13	26		
Protected Mode: Different Privilege Level														
Intersegment												61		
Intersegment Adding Immediate to SP												61		
ROL *Rotate Left*		x	-	-	-	-	-	-	-	x			b	h
Register/Memory by 1	D[000w] [mod 000 r/m]										2/4	2/4		
Register/Memory by CL	D[001w] [mod 000 r/m]										3/5	3/5		
Register/Memory by Immediate	C[000w] [mod 000 r/m] #										2/4	2/4		
ROR *Rotate Right*		x	-	-	-	-	-	-	-	x			b	h
Register/Memory by 1	D[000w] [mod 001 r/m]										2/4	2/4		
Register/Memory by CL	D[001w] [mod 001 r/m]										3/5	3/5		
Register/Memory by Immediate	C[000w] [mod 001 r/m] #										2/4	2/4		
RSDC *Restore Segment Register and Descriptor*	0F 79 [mod sreg3 r/m]	-	-	-	-	-	-	-	-	-	10	10	s	s
RSLDT *Restore LDTR and Descriptor*	0F 7B [mod 000 r/m]	-	-	-	-	-	-	-	-	-	10	10	s	s
RSM *Resume from SMM Mode*	0F AA	-	-	-	-	-	-	-	-	-	76	76	s	s
RSTS *Restore TSR and Descriptor*	0F 7D [mod 000 r/m]	-	-	-	-	-	-	-	-	-	10	10	s	s
SAHF *Store AH in FLAGS*	9E	x	-	-	-	x	x	-	x	x	2	2		
SAL *Shift Left Arithmetic*		x	-	-	-	x	x	-	x	x				
Register/Memory by 1	D[000w] [mod 100 r/m]										2/4	2/4		
Register/Memory by CL	D[001w] [mod 100 r/m]										3/5	3/5		
Register/Memory by Immediate	C[000w] [mod 100 r/m] #										2/4	2/4		
SAR *Shift Right Arithmetic*		x	-	-	-	x	x	x	x	x				
Register/Memory by 1	D[000w] [mod 111 r/m]										2/4	2/4		
Register/Memory by CL	D[001w] [mod 111 r/m]										3/5	3/5		
Register/Memory by Immediate	C[000w] [mod 111 r/m] #										2/4	2/4		
SBB *Integer Subtract with Borrow*		x	-	-	-	x	x	x	x	x			b	h
Register to Register	1[10dw] [11 reg r/m]										1	1		
Register to Memory	1[100w] [mod reg r/m]										3	3		
Memory to Register	1[101w] [mod reg r/m]										3	3		
Immediate to Register/Memory	8[00sw] [mod 001 r/m] #										1/3	1/3		
Immediate to Accumulator (short form)	1[110w] #										1	1		

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full) - = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS									REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF	DF	IF	TF	SF	ZF	AF	PF	C	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
SCAS *Scan String*	A [111w]	x	-	-	-	x	x	x	x	x	5		b	h
SETB/SETNAE/SETC *Set Byte on Below/Not Above or Equal/Carry* To Register/Memory	0F 92 [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETBE/SETNA *Set Byte on Below or Equal/Not Above* To Register/Memory	0F 96 [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETE/SETZ *Set Byte on Equal/Zero To Register/Memory*	0F 94 [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETL/SETNGE *Set Byte on Less/Not Greater or Equal* To Register/Memory	0F 9C [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETLE/SETNG *Set Byte on Less or Equal/Not Greater* To Register/Memory	0F 9E [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETNB/SETAE/SETNC *Set Byte on Not Below/ Above or Equal/Not Carry* To Register/Memory	0F 93 [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETNBE/SETA *Set Byte on Not Below or Equal/Above* To Register/Memory	0F 97 [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETNE/SETNZ *Set Byte on Not Equal/Not Zero* To Register/Memory	0F 95 [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETNL/SETGE *Set Byte on Not Less/Greater or Equal* To Register/Memory	0F 9D [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETNLE/SETG *Set Byte on Not Less or Equal/Greater* To Register/Memory	0F 9F [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETNO *Set Byte on Not Overflow* To Register/Memory	0F 91 [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETNP/SETPO *Set Byte on Not* Parity/Parity Odd To Register/Memory	0F 9B [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETNS *Set Byte on Not Sign* To Register/Memory	0F 99 [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETO *Set Byte on Overflow* To Register/Memory	0F 90 [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h
SETP/SETPE *Set Byte on Parity/Parity Even* To Register/Memory	0F 9A [mod 000 r/m]	-	-	-	-	-	-	-	-	-	2/2	2/2		h

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full) - = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS									REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF	DF	IF	TF	SF	ZF	AF	PF	C	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
SETS *Set Byte on Sign*		-	-	-	-	-	-	-	-	-				h
To Register/Memory	0F 98 [mod 000 r/m]										2/2	2/2		
SGDT *Store GDT Register*		-	-	-	-	-	-	-	-	-			b,c	h
To Register/Memory	0F 01 [mod 000 r/m]										6	6		
SHL *Shift Left Logical*		x	-	-	-	x	x	-	x	x			b	h
Register/Memory by 1	D [000w] [mod 100 r/m]										1/3	1/3		
Register/Memory by CL	D [001w] [mod 100 r/m]										2/4	2/4		
Register/Memory by Immediate	C [000w] [mod 100 r/m] #										1/3	1/3		
SHLD *Shift Left Double*		-	-	-	-	x	x	-	x	x				
Register/Memory by Immediate	0F A4 [mod reg r/m] #										1/3	1/3		
Register/Memory by CL	0F A5 [mod reg r/m]										3/5	3/5		
SHR *Shift Right Logical*		x	-	-	-	x	x	-	x	x			b	h
Register/Memory by 1	D [000w] [mod 101 r/m]										1/3	1/3		
Register/Memory by CL	D [001w] [mod 101 r/m]										2/4	2/4		
Register/Memory by Immediate	C [000w] [mod 101 r/m] #										1/3	1/3		
SHRD *Shift Right Double*		-	-	-	-	x	x	-	x	x				
Register/Memory by Immediate	0F AC [mod reg r/m] #										1/3	1/3		
Register/Memory by CL	0F AD [mod reg r/m]										3/5	3/5		
SIDT *Store IDT Register*		-	-	-	-	-	-	-	-	-			b,c	h
To Register/Memory	0F 01 [mod 001 r/m]										6	6		
SLDT *Store LDT Register*		-	-	-	-	-	-	-	-	-			a	h
To Register/Memory	0F 01 [mod 000 r/m]											1/2		
SMINT *Software SMM Entry*	0F 7E	-	-	-	-	-	-	-	-	-	24	24	s	s
SMSW *Store Machine Status Word*	0F 01 [mod 100 r/m]	-	-	-	-	-	-	-	-	-	1/2	1/2	b,c	h
STC *Set Carry Flag*	F9	-	-	-	-	-	-	-	-	1	1	1		
STD *Set Direction Flag*	FD	-	1	-	-	-	-	-	-	-	1	1		
STI *Set Interrupt Flag*	FB	-	-	1	-	-	-	-	-	-	7	7		m
STOS *Store String*	A [101w]	-	-	-	-	-	-	-	-	-	3	3	b	h
STR *Store Task Register*		-	-	-	-	-	-	-	-	-			a	h
To Register/Memory	0F 00 [mod 001 r/m]											1/2		
SUB *Integer Subtract*		x	-	-	-	x	x	x	x	x			b	h
Register to Register	2 [10dw] [11 reg r/m]										1	1		
Register to Memory	2 [100w] [mod reg r/m]										3	3		
Memory to Register	2 [101w] [mod reg r/m]										3	3		
Immediate to Register/Memory	8 [00sw] [mod 101 r/m] #										1/3	1/3		
Immediate to Accumulator (short form)	2 [110w] #										1	1		
SVDC *Save Segment Register and Descriptor*	0F 78 [mod sreg3 r/m]	-	-	-	-	-	-	-	-	-	18	18	s	s
SVLDT *Save LDTR and Descriptor*	0F 7A [mod 000 r/m]	-	-	-	-	-	-	-	-	-	18	18	s	s

= immediate data ++ = 16-bit displacement x = modified
+ = 8-bit displacement +++ = 32 bit displacement (full) - = unchanged

Table 6 - 17. Instruction Set Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF C	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
SVTS *Save TSR and Descriptor*	0F 7C [mod 000 r/m]	- - - - - - - - -	18	18	s	s
TEST *Test Bits*		0 - - - x x - x 0			b	h
Register/Memory and Register	8 [010w] [mod reg r/m]		1/3	1/3		
Immediate Data and Register/Memory	F [011w] [mod 000 r/m] #		1/3	1/3		
Immediate Data and Accumulator	A [100w] #		1	1		
VERR *Verify Read Access*		- - - - - x - - -			a	g,h,j,p
To Register/Memory	0F 00 [mod 100 r/m]			9/10		
VERW *Verify Write Access*		- - - - - x - - -			a	g,h,j,p
To Register/Memory	0F 00 [mod 101 r/m]			9/10		
WAIT *Wait Until FPU Not Busy*	9B	- - - - - - - - -	5	5		
WBINVD *Write-Back and Invalidate Cache*	0F 09	- - - - - - - - -	4	4		
XADD *Exchange and Add*		x - - - x x x x x				
Register1, Register2	0F C[000w] [11 reg2 reg1]		3	3		
Memory, Register	0F C[000w] [mod reg r/m]		6	6		
XCHG *Exchange*		- - - - - - - - -			b,f	f,h
Register/Memory with Register	8[011w] [mod reg r/m]		3/4	3/4		
Register with Accumulator	9[0 reg]		3	3		
XLAT *Translate Byte*	D7	- - - - - - - - -	3	3		h
XOR *Boolean Exclusive OR*		0 - - - x x - x 0			b	h
Register to Register	3 [00dw] [11 reg r/m]		1	1		
Register to Memory	3 [000w] [mod reg r/m]		3	3		
Memory to Register	3 [001w] [mod reg r/m]		3	3		
Immediate to Register/Memory	8 [00sw] [mod 110 r/m] #		1/3	1/3		
Immediate to Accumulator (short form)	3 [010w] #		1	1		

Instruction Notes for Instruction Set Summary

Notes a through c apply to Real Address Mode only:

a. This is a Protected Mode instruction. Attempted execution in Real Mode will result in exception 6 (invalid op-code).

b. Exception 13 fault (general protection) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum CS, DS, ES, FS, or GS segment limit (FFFFH). Exception 12 fault (stack segment limit violation or not present) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.

c. This instruction may be executed in Real Mode. In Real Mode, its purpose is primarily to initialize the CPU for Protected Mode.

Notes e through g apply to Real Address Mode and Protected Virtual Address Mode:

e. An exception may occur, depending on the value of the operand.
f. LOCK# is automatically asserted, regardless of the presence or absence of the LOCK prefix.
g. LOCK# is asserted during descriptor table accesses.

Notes h through r apply to Protected Virtual Address Mode only:

h. Exception 13 fault will occur if the memory operand in CS, DS, ES, FS, or GS cannot be used due to either a segment limit violation or an access rights violation. If a stack limit is violated, an exception 12 occurs.
i. For segment load operations, the CPL, RPL, and DPL must agree with the privilege rules to avoid an exception 13 fault. The segment's descriptor must indicate "present" or exception 11 (CS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 occurs.
j. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert LOCK# to maintain descriptor integrity in multiprocessor systems.
k. JMP, CALL, INT, RET, and IRET instructions referring to another code segment will cause an exception 13, if an applicable privilege rule is violated.
l. An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).
m. An exception 13 fault occurs if CPL is greater than IOPL.
n. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL and VM fields of the flag register are updated only if CPL = 0.
o. The PE bit of the MSW (CR0) cannot be reset by this instruction. Use MOV into CRO if desiring to reset the PE bit.
p. Any violation of privilege rules as apply to the selector operand does not cause a Protection exception, rather, the zero flag is cleared.
q. If the coprocessor's memory operand violates a segment limit or segment access rights, an exception 13 fault will occur before the ESC instruction is executed. An exception 12 fault will occur if the stack limit is violated by the operand's starting address.
r. The destination of a JMP, CALL, INT, RET, or IRET must be in the defined limit of a code segment or an exception 13 fault will occur.

Note s applies to SGS THOMSON specific SMM instructions:

s. All memory accesses to SMM space are non-cacheable. An invalid opcode exception 6 occurs unless SMI is enabled and SMAR size 0, and CPL = 0 and [SMAC is set or if in an SMI handler].

6.5 FPU Clock Counts

The CPU can be divided into the FPU which processes floating point instructions and the remaining circuity collectively called the integer unit. The FPU can execute instructions independently of the integer unit. For example, the integer unit can issue a floating point instruction without memory operands, in two clock cycles and then pass the operation to the FPU to execute. The integer unit will continue to execute instructions until the next floating point instruction is encountered. The FPU loads from memory are similar in that the integer unit issues the FPU instruction, transfers data to the FPU and then is free to execute integer instructions. However, when executing a floating point store, the resources of both the FPU and integer unit are used.

6.5.1 Instruction Set Summary

Table 6-18 summarizes the operation and allowed forms of the ST486DX/DX2 FPU instruction set.

6.5.2 Abbreviations

The abbreviations used in Table 6-18 are listed in the table below:

Table 6 - 18. FPU Table Abbreviations

ABBREVIATION	MEANING
n	Stack register number
TOS	Top of stack register pointed to by SSS in the status register.
ST(1)	FPU register next to TOS
ST(n)	A specific FPU register, relative to TOS
M.WI	16-bit integer operand from memory
M.SI	32-bit integer operand from memory
M.LI	64-bit integer operand from memory
M.SR	32-bit real operand from memory
M.DR	64-bit real operand from memory
M.XR	80-bit real operand from memory
M.BCD	18-digit BCD integer operand from memory
CC	FPU condition code
Env Regs	Status, Mode Control and Tag Registers, Instruction Pointer and Operand Pointer

Table 6 - 19. FPU Instruction Set Summary

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
F2XM1 *Function Evaluation 2x-1*	D9 F0	TOS <--- 2^{TOS}-1	98 - 114	See note 2
FABS *Floating Absolute Value*	D9 E1	TOS <--- | TOS |	5	
FADD *Floating Point Add*				
Top of Stack	DC [1100 0 n]	ST(n) <--- ST(n) + TOS	10 - 16	
80-bit Register	D8 [1100 0 n]	TOS <--- TOS + ST(n)	10 - 16	
64-bit Real	DC [mod 000 r/m]	TOS <--- TOS + M.DR	11 - 17	
32-bit Real	D8 [mod 000 r/m]	TOS <--- TOS + M.SR	13 - 19	
FADDP *Floating Point Add, Pop*	DE [1100 0 n]	ST(n) <--- ST(n) + TOS; then pop TOS	10 - 16	
FIADD *Floating Point Integer Add*				
32-bit integer	DA [mod 000 r/m]	TOS <--- TOS + M.SI	18 - 27	
16-bit integer	DE [mod 000 r/m]	TOS <--- TOS + M.WI	18 - 26	
FCHS *Floating Change Sign*	D9 E0	TOS <--- - TOS	5	
FCLEX *Clear Exceptions*	(9B) DB E2	Wait then Clear Exceptions	8	
FNCLEX *Clear Exceptions*	DB E2	Clear Exceptions	5	
FCOM *Floating Point Compare*				
80-bit Register	D8 [1101 0 n]	CC set by TOS - ST(n)	8	
64-bit Real	DC [mod 010 r/m]	CC set by TOS - M.DR	12	
32-bit Real	D8 [mod 010 r/m]	CC set by TOS - M.SR	10	
FCOMP *Floating Point Compare, Pop*				
80-bit Register	D8 [1101 1 n]	CC set by TOS - ST(n); then pop TOS	8	
64-bit Real	DC [mod 011 r/m]	CC set by TOS - M.DR; then pop TOS	12	
32-bit Real	D8 [mod 011 r/m]	CC set by TOS - M.SR; then pop TOS	10	
FCOMPP *Floating Point Compare, Pop* Two Stack Elements	DE D9	CC set by TOS - ST(1); then pop TOS and ST(1)	8	
FICOM *Floating Point Compare*				
32-bit integer	DA [mod 010 r/m]	CC set by TOS - M.WI	15 - 17	
16-bit integer	DE [mod 010 r/m]	CC set by TOS - M.SI	15 - 16	
FICOMP *Floating Point Compare*				
32-bit integer	DA [mod 011 r/m]	CC set by TOS - M.WI; then pop TOS	15 - 17	
16-bit integer	DE [mod 011 r/m]	CC set by TOS - M.SI; then pop TOS	15 - 16	
FCOS *Function Evaluation: Cos(x)*	D9 FF	TOS <--- COS(TOS)	98 - 143	See Note 1
FDECSTP *Decrement Stack Pointer*	D9 F6	Decrement top of stack pointer	5	
FDIV *Floating Point Divide*				
Top of Stack	DC [1111 1 n]	ST(n) <--- ST(n) / TOS	28 - 34	
80-bit Register	D8 [1111 0 n]	TOS <--- TOS / ST(n)	28 - 34	
64-bit Real	DC [mod 110 r/m]	TOS <--- TOS / M.DR	35 - 41	
32-bit Real	D8 [mod 110 r/m]	TOS <--- TOS / M.SR	33 - 39	
FDIVP *Floating Point Divide, Pop*	DE [1111 1 n]	ST(n) <--- ST(n) / TOS; then pop TOS	28 - 34	
FDIVR *Floating Point Divide Reversed*				
Top of Stack	DC [1111 0 n]	TOS <--- ST(n) / TOS	28 - 34	
80-bit Register	D8 [1111 1 n]	ST(n) <--- TOS / ST(n)	28 - 34	
64-bit Real	DC [mod 111 r/m]	TOS <--- M.DR / TOS	35 - 41	
32-bit Real	D8 [mod 111 r/m]	TOS <--- M.SR / TOS	33 - 39	

Table 6 - 19. FPU Instruction Set Summary (Continued)

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
FIDIVRP *Floating Point Integer Divide* Reversed, Pop	DE [1111 0 n]	ST(n) <--- TOS / ST(n); then pop TOS	28 - 34	
FIDIV *Floating Point Integer Divide*				
32-bit Integer				
16-bit Integer	DA [mod 110 r/m]	TOS <--- TOS / M.SI	36 - 44	
FIDIVR *Floating Point Integer* Reversed	DE [mod 110 r/m]	TOS <--- TOS / M.WI	36 - 43	
32-bit Integer	DA [mod 111 r/m]	TOS <--- M.SI / TOS	36 - 44	
16-bit Integer	DE [mod 111 r/m]	TOS <--- M.WI / TOS	36 - 43	
FFREE *Free Floating Point Register*	DD [1100 0 n]	TAG(n) <--- Empty	5	
FINCSTP *Increment Stack Pointer*	D9 F7	Increment top of stack pointer	5	
FINIT *Initialize FPU*	(9B)DB E3	Wait then initialize	8	
FNINIT *Initialize FPU*	DB E3	Initialize	5	
FLD *Load Data to FPU Reg.*				
Top of Stack	D9 [1100 0 n]	Push ST(n) onto stack	4	
80-bit Real	DB [mod 101 r/m]	Push M.XR onto stack	9	
64-bit Real	DD [mod 000 r/m]	Push M.DR onto stack	7	
32-bit Real	D9 [mod 000 r/m]	Push M.SR onto stack	5	
FBLD *Load Packed BCD Data to FPU Reg.*	DF [mod 100 r/m]	Push M.BCD onto stack	49 - 53	
FILD *Load Integer Data to FPU Reg.*				
64-bit Integer	DF [mod 101 r/m]	Push M.LI onto stack	9 - 13	
32-bit Integer	DB [mod 000 r/m]	Push M.SI onto stack	8 - 10	
16-bit Integer	DF [mod 000 r/m]	Push M.WI onto stack	8 - 9	
FLD1 *Load Floating Const. = 1.0*	D9 E8	Push 1.0 onto stack	6	
FLDCW *Load FPU Mode Control Register*	D9 [mod 101 r/m]	Ctl Word <--- Memory	5	
FLDENV *Load FPU Environment*	D9 [mod 100 r/m]	Env Regs <--- Memory	28 - 38	
FLDL2E *Load Floating Const. = Log2(e)*	D9 EA	Push $Log^2(e)$ onto stack	6	
FLDL2T *Load Floating Const. = Log2(10)*	D9 E9	Push $Log^2(10)$ onto stack	6	
FLDLG2 *Load Floating Const. = Log10(2)*	D9 EC	Push Log10(2) onto stack	6	
FLDLN2 *Load Floating Const. = Ln(2)*	D9 ED	Push Loge(2) onto stack	6	
FLDPI *Load Floating Const. = p*	D9 EB	Push π onto stack	6	
FLDZ *Load Floating Const. = 0.0*	D9 EE	Push 0.0 onto stack	6	
FMUL *Floating Point Multiply*				
Top of Stack	DC [1100 1 n]	ST(n) <--- ST(n) x TOS	12	
80-bit Register	D8 [1100 1 n]	TOS <--- TOS x ST(n)	12	
64-bit Real	DC [mod 001 r/m]	TOS <--- TOS x M.DR	15	
32-bit Real	D8 [mod 001 r/m]	TOS <--- TOS x M.SR	13	
FMULP *Floating Point Multiply & Pop*	DE [1100 1 n]	ST(n) <--- ST(n) x TOS; then pop TOS	12	
FIMUL *Floating Point Integer Multiply*				
32-bit Integer	DA [mod 001 r/m]	TOS <--- TOS x M.SI	21 - 25	
16-bit Integer	DE [mod 001 r/m]	TOS <--- TOS x M.WI	21 - 24	
FNOP *No Operation*	D9 D0	No Operation	3	

Table 6 - 17. Instruction Set Summary (Continued)

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
FPATAN *Function Eval: Tan^{-1} (y/x)*	D9 F3	ST(1) < --- ATAN[ST(1) / TOS]; then pop TOS	97 - 161	See Note 3
FPREM *Floating Point Reaminder*	D9 F8	TOS < --- Rem[TOS / ST(1)]	82 - 93	
FPREM1 *Floating Point Remainder IEEE*	D9 F5	TOS < --- Rem[TOS / ST(1)]	82 - 93	
FPTAN *Function Eval: Tax(x)*	D9 F2	TOS < --- TAN(TOS); then push 1.0 onto stack	123 - 140	See Note 1
FRNDINT *Round to Integer*	D9 FC	TOS < --- Round(TOS)	12 - 21	
FRSTOR *Load FPU Environment and Reg.*	DD [mod 100 r/m]	Restore state.	110 - 120	
FSAVE *Save FPU Environment and Reg*	(9B)DD [mod 110 r/m]	Wait then save state.	143 - 153	
FNSAVE *Save FPU Environment and Reg*	DD [mod 110 r/m]	Save state.	140 - 150	
FSCALE *Floating Multiply by2^{n}*	D9 FD	TOS < --- TOS Ñ 2$^{(ST(1))}$	10 - 15	
FSIN *Function Evaluation: Sin(x)*	D9 FE	TOS < --- SIN(TOS)	81 - 159	See Note 1
FSINCOS *Function Eval.: Sin(x) & Cos(x)*	D9 FB	temp < --- TOS; TOS < --- SIN(temp); then push COS(temp) onto stack	150 - 165	See Note 1
FSQRT *Floating Point Square Root*	D9 FA	TOS < --- Square Root of TOS	61 - 62	
FST *Store FPU Register*				
80-bit Register	DD [1101 0 n]	ST(n) < --- TOS	5	
80-bit Real	DB [mod 111 r/m]	M.XR < --- TOS	15	
64-bit Real	DD [mod 010 r/m]	M.DR < --- TOS	12	
32-bit Real	D9 [mod 010 r/m]	M.SR < --- TOS	9	
FSTP *Store FPU Register, Pop*				
Top of Stack	DB [1101 1 n]	ST(n) < --- TOS; then pop TOS	5	
80-bit Real	DB [mod 111 r/m]	M.XR < --- TOS; then pop TOS	15	
64-bit Real	DD [mod 011 r/m]	M.DR < --- TOS; then pop TOS	12	
32-bit Real	D9 [mod 011 r/m]	M.SR < --- TOS; then pop TOS	9	
FBSTP *Store BCD Data, Pop*	DF [mod 110 r/m]	M.BCD < --- TOS; then pop TOS	77 - 82	
FIST *Store Integer FPU Register*				
32-bit Integer	DB [mod 010 r/m]	M.SI < --- TOS	16 - 22	
16-bit Integer	DF [mod 010 r/m]	M.WI < --- TOS	12 - 18	
FISTP *Store Integer FPU Register, Pop*				
64-bit Integer	DF [mod 111 r/m]	M.LI < --- TOS; then pop TOS	19 - 27	
32-bit Integer	DB [mod 011 r/m]	M.SI < --- TOS; then pop TOS	16 - 22	
16-bit Integer	DF [mod 011 r/m]	M.WI < --- TOS; then pop TOS	12 - 18	
FSTCW *Store FPU Mode Control Register*	(9B)D9 [mod 111 r/m]	Wait Memory < --- Control Mode Register	6	
FNSTCW *Store FPU Mode Control Register*	D9 [mod 111 r/m]	Memory < --- Control Mode Register	3	
FSTENV *Store FPU Environment*	(9B)D9 [mod 110 r/m]	Wait Memory < --- Env. Registers	30 - 40	
FNSTENV *Store FPU Environment*	D9 [mod 110 r/m]	Memory < --- Env. Registers	27 - 37	
FSTSW *Store FPU Status Register*	(9B)DD [mod 111 r/m]	Wait Memory < --- Status Register	6	
FNSTSW *Store FPU Status Register*	DD [mod 111 r/m]	Memory < --- Status Register	3	
FSTSW AX *Store FPU Status Register to AX*	E0	Wait AX < --- Status Register	6	
FNSTSW AX *Store FPU Status Register to AX*	E0	AX < --- Status Register	3	

Table 6 - 17. Instruction Set Summary (Continued)

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
FSUB *Floating Point Subtract*				
Top of Stack	DC [1110 1 n]	ST(n) <--- ST(n) - TOS	10 - 16	
80-bit Register	D8 [1110 0 n]	TOS <--- TOS - ST(n)	10 - 16	
64-bit Real	DC [mod 100 r/m]	TOS <--- TOS - M.DR	13 - 19	
32-bit Real	D8 [mod 100 r/m]	TOS <--- TOS - M.SR	11 - 17	
FSUBP *Floating Point Subtract, Pop*	DE [1110 1 n]	ST(n) <--- ST(n) - TOS; then pop TOS	10 - 16	
FSUBR *Floating Point Subtract Reverse*				
Top of Stack	DC [1110 0 n]	TOS <--- ST(n) - TOS	10 - 16	
80-bit Register	D8 [1110 1 n]	ST(n) <--- TOS - ST(n)	10 - 16	
64-bit Real	DC [mod 101 r/m]	TOS <--- M.DR - TOS	13 - 19	
32-bit Real	D8 [mod 101 r/m]	TOS <--- M.SR - TOS	11 - 17	
FSUBRP *Floating Point Subtract Reverse, Pop*	DE [11100 n]	ST(n) <--- TOS - ST(n); then pop TOS	10 - 16	
FISUB *Floating Point Integer Subtract*				
32-bit Integer	DA [mod 100 r/m]	TOS <--- TOS - M.SI	18 - 27	
16-bit Integer	DE [mod 100 r/m]	TOS <--- TOS - M.WI	18 - 26	
FISUBR *Floating Point Integer Subtract Reverse*				
32-bit Integer Reversed	DA [mod 101 r/m]	TOS <--- M.SI - TOS	18 - 27	
16-bit Integer Reversed	DE [mod 101 r/m]	TOS <--- M.WI - TOS	18 - 26	
FTST *Test Top of Stack*	D9 E4	CC set by TOS - 0.0	10	
FUCOM *Unordered Compare*	DD [1110 0 n]	CC set by TOS - ST(n)	8	
FUCOMP *Unordered Compare, Pop*	DD [1110 1 n]	CC set by TOS - ST(n); then pop TOS	8	
FUCOMPP *Unordered Compare, Pop two elements*	DA E9	CC set by TOS - ST(I); then pop TOS and ST(1)	8	
FWAIT *Wait*	9B	Wait for FPU not busy	3	
FXAM *Report Class of Operand*	D9 E5	CC <--- Class of TOS	4	
FXCH *Exchange Register with TOS*	D9 [1100 1 n]	TOS <--- ST(n) Exchange	9	
FXTRACT *Extract Exponent*	D9 F4	temp <--- TOS; TOS <--- exponent (temp); then push significant (temp) onto stack	11 - 16	
FLY2X *Function Eval. y x $Log_2(x)$*	D9 F1	ST(1) <--- ST(1) x Log_2(TOS); then pop TOS	145 - 154	
FLY2XP1 *Function Eval. y x $Log_2(x+1)$*	D9 F9	ST(1) <--- ST(1) x Log_2(1+TOS); then pop TOS	131 - 133	See Note 4

FPU Instruction Summary Notes

All references to TOS and ST(n) refer to stack layout prior to execution.

Values popped off the stack are discarded.

A pop from the stack increments the top stack pointer.

A push to the stack decrements the top of the stack pointer.

Note 1:
For FCOS, FSIN, FSINCOS and FPTAN, time shown is for absolute value of TOS < $3\pi/4$.
Add 90 clock counts for argument reduction if outside this range.

For FCOS, clock count is 143 if TOS < $\pi/4$ and clock count is 98 if $\pi/4$ < TOS > $\pi/2$
For FSIN, clock count is 81 to 82 if absolute value of TOS < $\pi/4$.

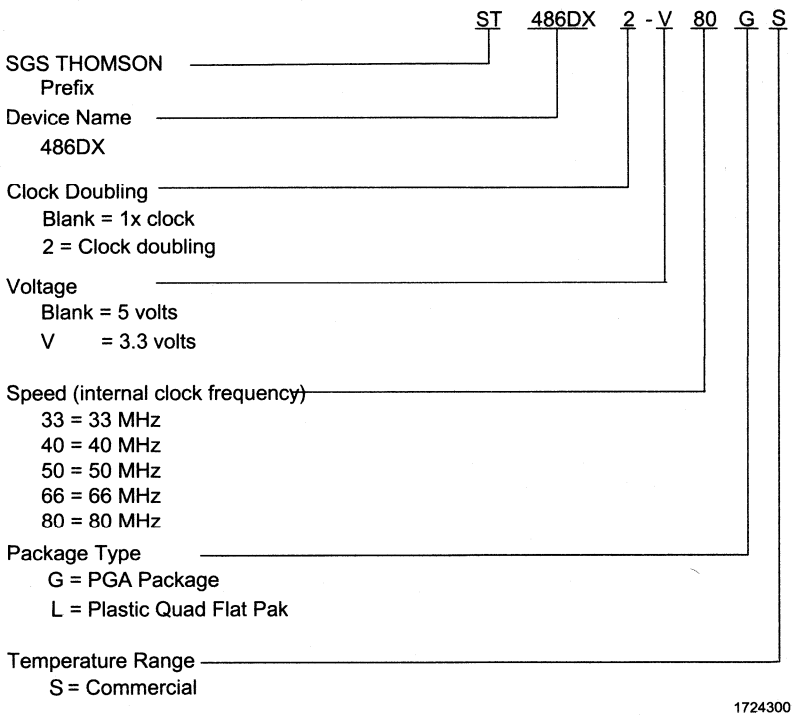
Note 2:
For F2XM1, clock count is 98 if absolute value of TOS < 0.5.

Note 3:
For FPATAN, clock count is 97 if ST(1)/TOS < $\pi/32$.

Note 4:
For FYL2XP1, clock count is 170 if TOS is out of range and regular FYL2X is called.

INDEX ORDERING INFORMATION

Ordering Information



The ST486DX and ST486DX2 part numbers are listed below:

ST486DX and ST486DX2 Part Numbers

PART NUMBER	Vcc (V)	FREQUENCY (MHz)		PACKAGE		AC Specification
		BUS	INTERNAL	QFP	PGA	
ST486DX-33GS	5.0	33	33		x	Table 4-8
ST486DX-40GS	5.0	40	40		x	Table 4-9
ST486DX-50GS	5.0	50	50		x	Table 4-10
ST486DX2-50GS	5.0	25	50		x	Table 4-7
ST486DX2-66GS	5.0	33	66		x	Table 4-8
ST486DX-V33GS	3.3	33	33		x	Table 4-12
ST486DX-V33QS	3.3	33	33	x		Table 4-12
ST486DX-V40GS	3.3	40	40		x	Table 4-13
ST486DX-V40QS	3.3	40	40	x		Table 4-13
ST486DX2-V50GS	3.3	25	50		x	Table 4-11
ST486DX2-V50QS	3.3	25	50	x		Table 4-11
ST486DX2-V66GS	3.3	33	66		x	Table 4-12
ST486DX2-V66QS	3.3	33	66	x		Table 4-12
ST486DX2-V80GS	3.3	40	80		x	Table 4-13
ST486DX2-V80QS	3.3	40	80	x		Table 4-13

INDEX

INDEX

S

T

V

W

SALES OFFICES

EUROPE

DENMARK

2730 HERLEV
Herlev Torv, 4
Tel. (45-44) 94.85.33
Telex: 35411
Telefax: (45-44) 948694

FINLAND

LOHJA SF-08150
Ratakatu, 26
Tel. (358-12) 155.11
Telefax. (358-12) 155.66

FRANCE

94253 GENTILLY Cedex
7 - avenue Gallieni - BP. 93
Tel.: (33-1) 47.40.75.75
Telex: 632570 STMHQ
Telefax: (33-1) 47.40.79.10

67000 STRASBOURG
20, Place des Halles
Tel. (33-88) 75.50.66
Telefax: (33-88) 22.29.32

GERMANY

85630 GRASBRUNN
Bretonischer Ring 4
Postfach 1122
Tel.: (49-89) 460060
Telefax: (49-89) 4605454
Teletex: 897107=STDISTR

60327 FRANKFURT
Gutleutstrasse 322
Tel. (49-69) 237492-3
Telefax: (49-69) 231957
Teletex: 6997689=STVBF

30695 HANNOVER 51
Rotenburger Strasse 28A
Tel. (49-511) 615960-3
Teletex: 5118418 CSFBEH
Telefax: (49-511) 6151243

90491 NÜRNBERG 20
Erlenstegenstrasse, 72
Tel.: (49-911) 59893-0
Telefax: (49-911) 5980701

70499 STUTTGART 31
Mittlerer Pfad 2-4
Tel. (49-711) 13968-0
Telefax: (49-711) 8661427

ITALY

20090 ASSAGO (MI)
V.le Milanofiori - Strada 4 - Palazzo A/4/A
Tel. (39-2) 57546.1 (10 linee)
Telex: 330131 - 330141 SGSAGR
Telefax: (39-2) 8250449

40033 CASALECCHIO DI RENO (BO)
Via R. Fucini, 12
Tel. (39-51) 591914
Telex: 512442
Telefax: (39-51) 591305

00161 ROMA
Via A. Torlonia, 15
Tel. (39-6) 8553960
Telex: 620653 SGSATE I
Telefax: (39-6) 8444474

NETHERLANDS

5652 AR EINDHOVEN
Meerenakkerweg 1
Tel.: (31-40) 550015
Telex: 51186
Telefax: (31-40) 528835

SPAIN

08004 BARCELONA
Calle Gran Via Corts Catalanes, 322
6th Floor, 2th Door
Tel. (34-3) 4251800
Telefax: (34-3) 4253674

28027 MADRID
Calle Albacete, 5
Tel. (34-1) 4051615
Telex: 46033 TCCEE
Telefax: (34-1) 4031134

SWEDEN

S-16421 KISTA
Borgarfjordsgatan, 13 - Box 1094
Tel.: (46-8) 7939220
Telex: 12078 THSWS
Telefax: (46-8) 7504950

SWITZERLAND

1218 GRAND-SACONNEX (GENEVA)
Chemin Francois-Lehmann, 18/A
Tel. (41-22) 7986462
Telex: 415493 STM CH
Telefax: (41-22) 7984869

UNITED KINGDOM and EIRE

MARLOW, BUCKS
Planar House, Parkway
Globe Park
Tel.: (44-628) 890800
Telex: 847458
Telefax: (44-628) 890391

AMERICAS

BRAZIL

05413 SÃO PAULO
R. Henrique Schaumann 286-CJ33
Tel.: (55-11) 883-5455
Telex: (391)11-37988 "UMBR BR"
Telefax : (55-11) 282-2367

CANADA

NEPEAN ONTARIO K2H 9C4
301 Moodie Drive Suite 307
Tel.: (613) 829-9944
Telefax: (613) 829-8998

U.S.A.

NORTH & SOUTH AMERICAN
MARKETING HEADQUARTERS
55 Old Bedford Road
Lincoln, MA 01773
Tel.: (617) 259-0300
Telefax: (617) 259-4421

SALES COVERAGE BY STATE

ALABAMA
Huntsville - Tel.: (205) 533-5995
Fax: (205) 533-9320

ARIZONA
Phoenix - Tel.: (602) 867-6217
Fax: (602) 867-6200

CALIFORNIA
Santa Ana - Tel.: (714) 957-6018
Fax: (714) 957-3281
San Jose - Tel.: (408) 452-8585
Fax: (452) 1549

COLORADO
Boulder - Tel.: (303) 449-9000
Fax: (303) 449-9505

FLORIDA
Boca Raton - Tel.: (407) 997-7233
Fax: (407) 997-7554

GEORGIA
Norcross - Tel.: (404) 242-7444
Fax: (404) 368-9439

ILLINOIS
Schaumburg - Tel.: (708) 517-1890
Fax: (708) 517-1899

INDIANA
Kokomo - Tel.: (317) 455-3500
Fax: (317) 455-3400
Indianapolis - Tel.: (317) 575-5520
Fax: (317) 575-8211

MICHIGAN
Livonia - Tel.: (313) 953-1700
Fax: (313) 462-4071

MINNESOTA
Bloomington - Tel.: (612) 944-0098
Fax: (612) 944-0133

NORTH CAROLINA
Cary - Tel.: (919) 469-1311
Fax: (919) 469-4515

NEW JERSEY
Voorhees - Tel.: (609) 772-6222
Fax: (609) 772-6037

NEW YORK
Poughkeepsie - Tel.: (914) 454-8813
Fax: (914) 454-1320

OREGON
Lake Oswego - Tel.: (503) 635-7650

TENNESSEE
Knoxville - Tel.: (615) 524-6239

TEXAS
Austin - Tel.: (512) 502-3020
Fax: (512) 346-6260
Carrollton - Tel.: (214) 466-8844
Fax: (214) 466-8130
Houston - Tel.: (713) 376-9936
Fax: (713) 376-9948

FOR RF AND MICROWAVE
POWER TRANSISTORS CON-
TACT
THE FOLLOWING REGIONAL
OFFICE IN THE U.S.A.

PENNSYLVANIA
Montgomeryville - Tel.: (215) 361-6400
Fax: (215) 361-1293

ASIA / PACIFIC

AUSTRALIA

NSW 2220 HURTSVILLE
Suite 3, Level 7, Otis House
43 Bridge Street
Tel. (61-2) 5803811
Telefax: (61-2) 5806440

HONG KONG

WANCHAI
22nd Floor - Hopewell centre
183 Queen's Road East
Tel. (852) 8615788
Telex: 60955 ESGIES HX
Telefax: (852) 8656589

INDIA

NEW DELHI 110019
Liaison Office
3rd Floor, F-Block
International Trade Tower
Nehru Place
Tel. (91-11) 644-5928/647-9415
Telex: 031-70193 STMI IN
Telefax: (91-11) 6443054

MALAYSIA

SELANGOR, PETALING JAYA 46200
Unit BM-10
PJ Industrial Park
Jalan Kemajuan 12/18
Tel.: (03) 758 1189
Telefax: (03) 758 1179

PULAU PINANG 10400
4th Floor - Suite 4-03
Bangunan FOP-123D Jalan Anson
Tel. (04) 379735
Telefax (04) 379816

KOREA

SEOUL 121
8th floor Shinwon Building
823-14, Yuksam-Dong
Kang-Nam-Gu
Tel. (82-2) 553-0399
Telex: SGSKOR K29998
Telefax: (82-2) 552-1051

SINGAPORE

SINGAPORE 2056
28 Ang Mo Kio - Industrial Park 2
Tel. (65) 4821411
Telex: RS 55201 ESGIES
Telefax: (65) 4820240

TAIWAN

TAIPEI
11th Floor
105, Section 2 Tun Hua South Road
Tel. (886-2) 755-4111
Telex: 10310 ESGIE TW
Telefax: (886-2) 755-4008

THAILAND

BANGKOK 10110
54 Asoke Road
Sukhumvit 21
Tel.: (662) 260 7870
Telefax: (662) 260 7871

JAPAN

TOKYO 108
Nisseki - Takanawa Bld. 4F
2-18-10 Takanawa
Minato-Ku
Tel. (81-3) 3280-4121
Telefax: (81-3) 3280-4131