TEXAS INSTRUMENTS

TI486SXLC and TI486SXL Microprocessors

Reference Guide

1994

PC Systems Logic Products

Texas Instruments

Printed in U.S.A.
1094–IP

SRZU006D

Reference Guide

TI486SXLC and TI486SXL Microprocessors

1994

TEXAS INSTRUMENTS

TI486SXLC and TI486SXL Microprocessors

1994

PC Systems Logic Products

Texas Instruments

Printed in U.S.A.
1094–IP

SRZU006D

TI486SXLC and TI486SXL Microprocessors

1994

Reference Guide

Reference
Guide

[redacted]

[redacted]

[redacted]

TI486SXLC and TI486SXL Microprocessors

Reference Guide

October 1994

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Preface

Read This First

About This Manual

This manual describes the TI486SXL(C) microprocessor product family. Each chapter except for chapters 3 and 4 covers all versions of the TI486SXLC and the TI486SXL microprocessors. Chapter 3 explicitly covers the TI486SXLC series and chapter 4 explicitly covers the TI486SXL series.

How to Use This Manual

This document contains the following chapters:

Chapter 1 Product Overview

Chapter 1 introduces the features of the TI486SXLC and TI486SXL microprocessor series and defines the differences between them. A functional block diagram, logic symbol, and I/O signal identifications are provided for each of the two series of microprocessors. Additional material describes selected system architectures such as the execution pipeline, the on-chip cache memory, and the power-management techniques.

Chapter 2 Programming Interface

Chapter 2 describes the internal operations of the TI486SXL(C) family of microprocessors mainly from an application programmer's point of view. Included in this chapter are descriptions of processor initialization, the register sets, memory addressing, various types of interrupts, system-management mode, and the shutdown and halt process. Overviews of real, virtual-8086, and protected operating modes are also included.

Chapter 3 TI486SXLC Microprocessor Bus Interface

Chapter 3 provides a summary of the TI486SXLC series processor signals and descriptions of all inputs/outputs, functional timing and bus operations (including pipelined and nonpipelined addressing), various interfaces, and power management.

Chapter 4 TI486SXL Microprocessor Bus Interface

Chapter 4 provides a summary of the TI486SXL series processor signals and descriptions of all inputs/outputs, functional timing and bus operations (including pipelined and nonpipelined addressing), various interfaces, and power management.

Chapter 5 Electrical Specifications

Chapter 5 provides electrical specifications for the TI486SXL(C) family, including specifications for the 3.3-volt versions. The specifications include electrical connection requirements for all package pins, maximum ratings, recommended operating conditions, dc electrical characteristics, and ac characteristics.

Chapter 6 Mechanical Specifications

Chapter 6 provides mechanical specifications for the TI486SXL(C) family that include pin assignments, package physical dimensions, and package thermal characteristics.

Chapter 7 Instruction Set

Chapter 7 summarizes the instruction set for the TI486SXL(C) family and provides detailed information of the instruction encoding. The instruction set is the same for all TI486SXL(C) microprocessors. Instructions are listed in an instruction set summary table that provides information on the flags affected and the instruction clock counts for each instruction.

Appendix A Programming System Management Mode (SMM)

Appendix A provides detailed information including examples pertinent to programming the TI486SXL(C) system management mode (SMM). Included are implementing system-management interrupt (SMI), testing/debugging SMM code, utilizing power management, loading SMM programs, detecting the type of CPU, detecting the presence of SMM-capable devices, creating macros, and altering SMM code limits.

Appendix B BIOS Modifications Guide

Appendix B discusses types of BIOS changes to be considered by the PC designer. The topics are power-on and hard reset, protected-mode to real-mode switching, and soft reset. Examples of assembler code for turning the cache on and off are provided.

Appendix C Design Considerations and Cache Flush

Appendix C provides design considerations, address bit A20 masking, and general cache invalidation procedures.

Appendix D OEM Modifications for 168-Pin CPGA

Appendix D describes the potential modifications an OEM needs to implement on an existing 486SX/DX/DX4 motherboard to take advantage of the TI486SXL 168 pin ceramic pin grid array (CPGA). A system implementation is described for a 3.3-V system that supports a 5-V industry standard architecture (ISA) and a 3.3-V VL (VESA, Video Electronics Specifications Association local bus), and another implementation is described for a mixed 3.3-V/5-V system that supports a 5-V ISA and a 5-V VL bus.

Appendix E Thermal Management in Microprocessor-Based Systems

Appendix E provides the reader with basic thermal concepts and the relationship between thermal measurements and the system. In addition, problems associated with comparing thermal specifications from different manufacturers are discussed. Finally, corrective activity within JEDEC is detailed.

Appendix F Ordering Information

Appendix F provides detailed ordering information showing the meaning of the components of the part number and a description of each microprocessor offered.

Appendix G Glossary

Appendix G contains explanations for the terms, abbreviations, and acronyms used in this manual.

Notational Conventions

This document uses the following conventions.

- Program code listings and program code examples are shown in a `special typeface` similar to a typewriter's.

  Here is a sample assembler code program listing:

```
CLI
MOV    EAX, CR0       ; set bit 30, turn off cache
OR     EAX, 40000000h ; for external cache coherency
```

- In the instruction syntax descriptions, the instruction is in a **bold typeface** and a description of the instruction is in *italic typeface*. Here is an example of an instruction syntax and description:

  RSM *Resume from SMM Mode*

Information About Cautions and Warnings

This book may contain cautions and warnings.

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

This is an example of a warning statement.

A warning statement describes a situation that could potentially cause harm to you.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

Trademarks

AMD is a trademark of Advanced Micro Devices, Inc.

EPIC is a trademark of Texas Instruments Incorporated.

HP is a trademark of Hewlett-Packard Co.

Intel is a trademark of Intel Corporation.

Contents

Figures

Tables

Equations

Examples

Chapter 1

Product Overview

This chapter introduces the features of the TI486SXLC series and TI486SXL series of microprocessors and defines the differences between them. A functional block diagram, logic symbol, and I/O signal identifications are provided for the TI486SXLC and TI486SXL series of microprocessors. Additional material describes selected system architectures such as the execution pipeline, the on-chip cache memory, and the power-management techniques.

Topic **Page**

1.1 Features

The TI486SXLC and TI486SXL series microprocessors are attractive for new 486-compatible system designs as they are instruction-set and footprint compatible with existing platforms. Additionally, they implement high-performance levels, including clock-doubled CPUs with on-chip 8K-byte cache, advanced power-management techniques, and industry-standard pinouts that simplify implementation of energy-efficient desktop and/or battery-powered notebook systems. Their expanded features are:

- 486 architecture and performance
  - 486-compatible instruction set and register set
  - On-chip 8K-byte, 32-bit instruction/data cache configured as two-way set associative
  - Clock-doubled 3.3-V with 5-V tolerant I/Os and 5-V versions
  - Highly optimized, variable-length pipeline
  - On-chip 16-bit hardware multiplier
- High-performance, footprint-compatible upgrade path for existing TI486SLC and TI486DLC platforms
  - Clock speeds up to 50 MHz
  - Industry standard footprints:
    TI486SXLC series uses 100-pin QFP (486SLC footprint)
    TI486SXL series uses 132-pin PGA (486DLC footprint), 144-pin plastic or ceramic QFP (486DLC footprint), and a 168-pin CPGA (486SX footprint)
- Advanced power-management features for battery-powered notebook and energy-efficient desktop PC systems
  - System-management mode (SMM)
  - High-priority system-management interrupt (SMI) with separate memory-address space
  - Suspend mode (initiated by either hardware or software)
  - Dynamic clock scaling
  - Fully static device permits clock-stop state
  - Approximately 60-percent power savings in 3.3-V versions
  - 3.3-V versions with 5-V tolerant inputs and outputs (available in the TI486SXL series) can be used in 3.3-V-only or mixed 3.3-V/5-V systems

Features (Continued)

- ❑ Texas Instruments EPIC™ submicron CMOS technology
- ❑ 32-bit internal and 16-bit external buses in the TI486SXLC series. This product offering is shown in Table 1–1.

Table 1–1. TI486SXLC Product Offering

TI486SXLC Series Device Part Number	Supply Voltage (V)	Speed (MHz)		Package
		Core	Bus	
TX486SXLC-V25-PJF	3.3	25	25	100-pin QFP
TX486SXLC-040-PJF	5	40	40, 20†	
TX486SXLC2-050-PJF	5	50	25	

† These microprocessors can be operated as nonclock-doubled 40 MHz or clock-doubled 20/40 MHz.

- ❑ 32-bit internal and 32-bit external buses in the TI486SXL series. This product offering is shown in Table 1–2

Table 1–2. TI486SXL Product Offering

TI486SXL Series Device Part Number	Supply Voltage (V)	Speed (MHz)		Package
		Core	Bus	
TX486SXL-040S-GA	5	40	40, 20†	132-pin PGA
TX486SXL2-050S-GA	5	50	25	
TX486SXL-040-PCE	5	40	40, 20†	144-pin TEP
TX486SXL-G40-HBN	3.3-V, 5-V tolerant	40	40, 20†	144-pin ceramic QFP
TX486SXL2-G50-HBN	3.3-V, 5-V tolerant	50	25	
TX486SXL-040-HBN	5	40	40, 20†	
TX486SXL2-050-HBN	5	50	25	
TX486SXL-G40-GA	3.3-V, 5-V tolerant	40	40, 20†	168-pin PGA
TX486SXL2-G50-GA	3.3-V, 5-V tolerant	50	25	
TX486SXL-V40-GA	3.3	40	40, 20†	
TX486SXL2-V50-GA	3.3	50	25	
TX486SXL-040-GA	5	40	40, 20†	
TX486SXL2-050-GA	5	50	25	

† These microprocessors can be operated as nonclock-doubled 40 MHz or clock-doubled 20/40 MHz.

For an explanation of the part numbers see Appendix F.

1.2 Introduction

The Texas Instruments TI486SXL(C) microprocessor family is comprised of advanced x86-compatible processors that offer clock-doubled features for increased system performance. Each processor provides an internal 8K-byte, 32-bit cache and integrated power management on a single chip.

The fully static, 486-instruction-set-compatible TI486SXLC series microprocessors are backward compatible with the TI486SLC/E. The TI486SXLC2 microprocessors contain a clock-doubled feature for increased system performance up to 50 MHz. The TI486SXLC series is an ideal solution for battery-powered applications as it typically draws only 0.1-mA supply current while the input clock is stopped in suspend mode. The TI486SXLC-V25 offers additional power savings as it operates from a 3.3-V power supply.

The fully static, 486-instruction-set-compatible TI486SXL series microprocessors are available in three package types: a 132-pin PGA, 144-pin QFP, and a 168-pin PGA. The 132-pin PGA TI486SXL and TI486SXL2 are backward compatible with the TI486DLC/E. The 144-pin QFP TI486SXL and TI486SXL2 are backward compatible with the 486DLC footprint. The 168-pin PGA TI486SXL and TI486SXL2 have the same footprint as the 486SX pinout (see Appendix D, *OEM Modifications for 168-Pin CPGA*). The TI486SXL2 microprocessors contain a clock-doubled feature for increased system performance up to 50 MHz. The TI486SXL series is an ideal solution for battery-powered applications as it typically draws only 0.1 mA while the input clock is stopped in suspend mode. The TI486SXL-V40 and TI486SXL2-V50 offer additional power savings as they operate from a 3.3-V power supply. The TI486SXL-G40 and TI486SXL2-G50 offer the equivalent power savings with the added capability to operate in either 3.3-V-only systems or in mixed 3.3-V/5-V systems.

The TI486SXL series microprocessors support 8-, 16-, and 32-bit data types and operate in real, virtual-8086, and protected modes. The TI486SXL(C) microprocessor family achieves high performance through use of a highly optimized, variable-length pipeline combined with a RISC-like, single-cycle execution unit, an on-chip hardware multiplier, and an 8K-byte integrated instruction and data cache.

1.3 TI486SXLC Series Overview

The TI486SXLC series of microprocessors are implemented using Texas Instruments EPIC™ submicron CMOS technology. The combination of high-performance 486-like operation, internal 8K-byte cache, advanced power management, and small-form-factor package makes the TI486SXLC series ideal for notebook/subnotebook applications.

A summary of the product offering is shown in Table 1–3. Figure 1–1 is a functional block diagram and Figure 1–2 is the logic symbol for the TI486SXLC microprocessors.

Table 1–3. TI486SXLC Microprocessors

Device	Supply Voltage (V)	Speed (MHz)		Package†
		Core	Bus	
TI486SXLC-V25	3.3	25	25	100-pin QFP
TI486SXLC-040	5	40	40, 20‡	
TI486SXLC2-050	5	50	25	

† Pinout and footprint compatible with TI486SLC/E.
‡ These microprocessors can be operated as nonclock-doubled 40 MHz or clock-doubled 20/40 MHz.

Figure 1–1. TI486SXLC Functional Block Diagram

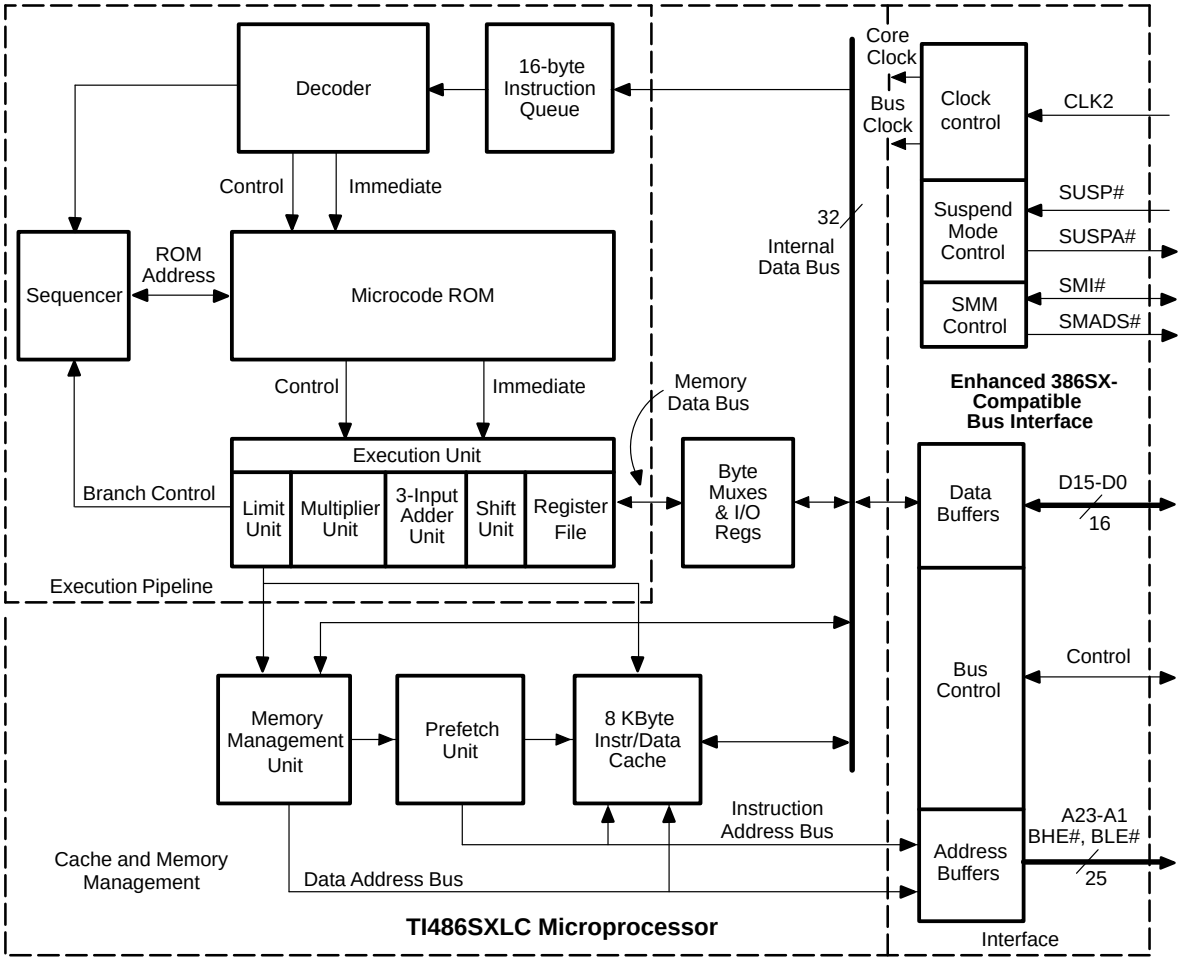
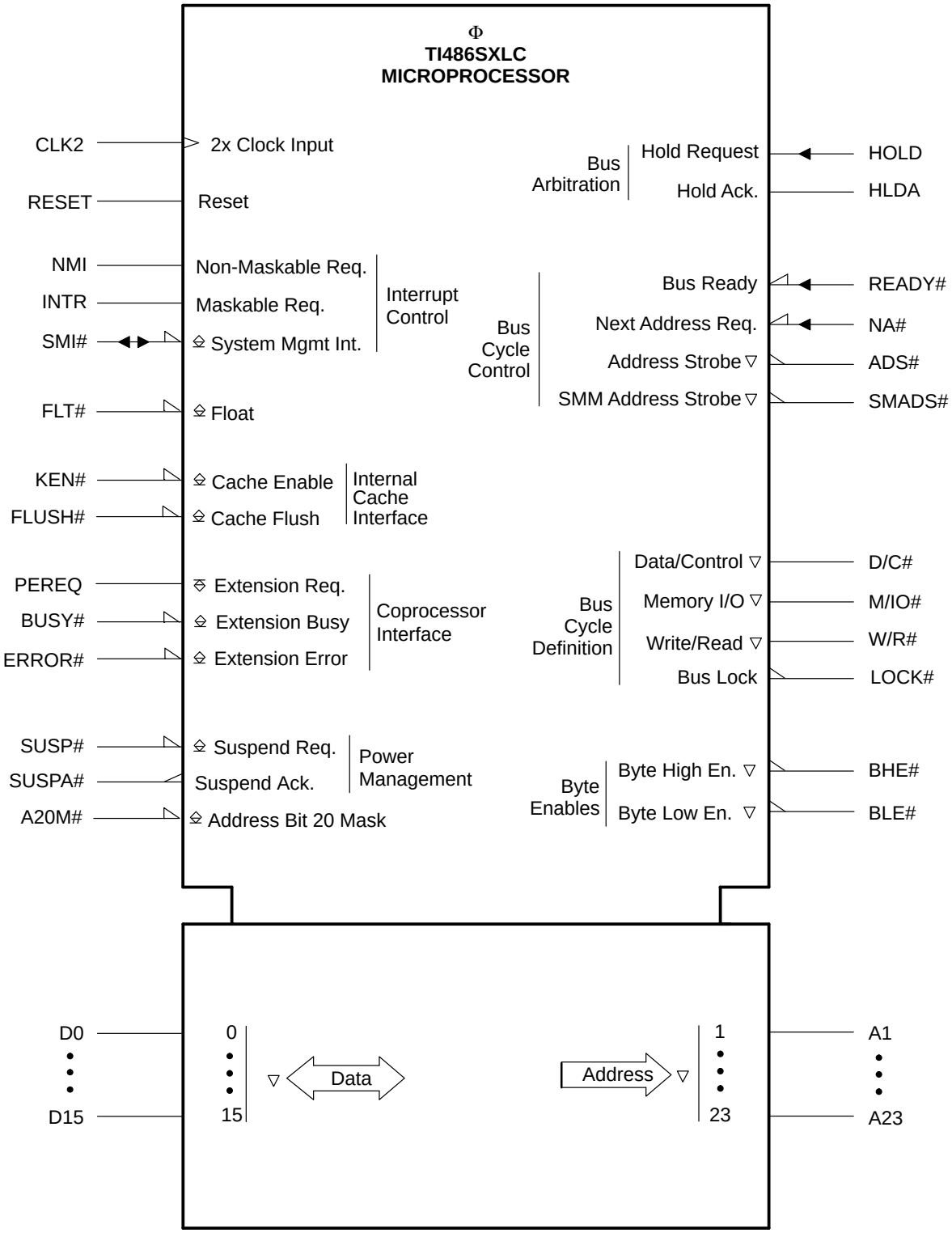


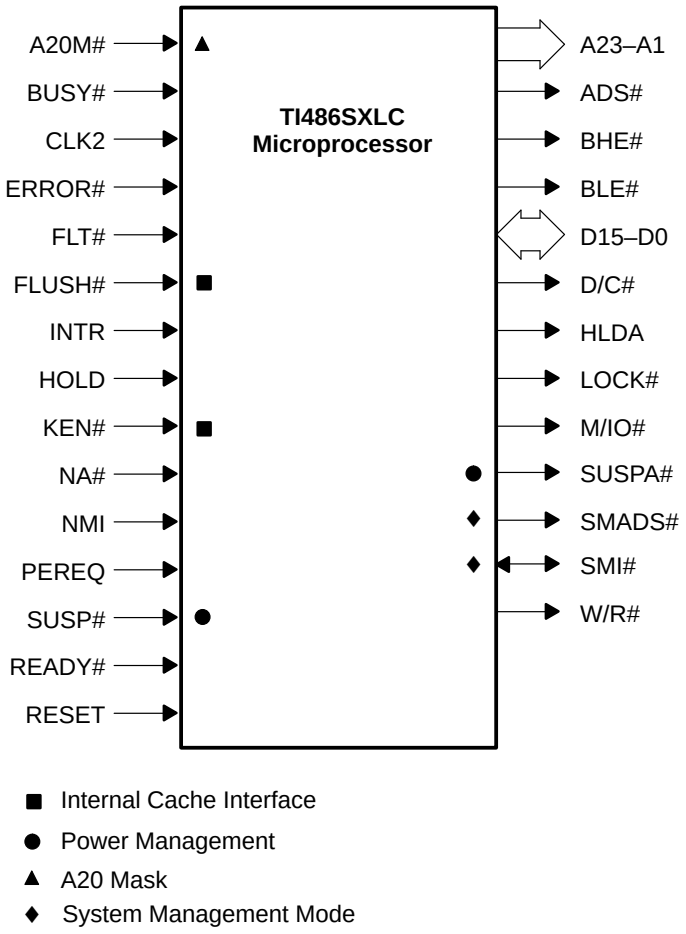
Figure 1–2. TI486SXLC Logic Symbol†



† This symbol is in accordance with ANSI/IEEE Std 91-1991 and IEC Publication 617-12.

The TI486SXLC includes two power-management signals (SUSP# and SUSPA#), two cache-interface signals (FLUSH# and KEN#), an A20 mask input (A20M#), and two SMM signals (SMADS# and SMI#) that are additions to the 386SX signal set. The TI486SXLC series has the same signal set as the TI486SLC/E microprocessor. The complete list of TI486SXLC signals is shown in Figure 1–3.

Figure 1–3. TI486SXLC Input and Output Signals



1.4 TI486SXL Series Overview

The TI486SXL series of microprocessors are implemented using Texas Instruments EPIC submicron CMOS technology. The combination of high-performance 486-like operation, internal 8K-byte cache, 32-bit external data path, and advanced power-management features makes the TI486SXL series ideal for energy-efficient desktop and notebook applications.

A summary of the product offering is shown in Table 1–4. Figure 1–4 is a functional block diagram and Figure 1–5 and Figure 1–6 are logic symbols for the 132-pin, 144-pin, and 168-pin TI486SXL microprocessors.

Table 1–4. TI486SXL Microprocessors

Device	Supply Voltage (V)	Speed (MHz)		Package
		Core	Bus	
TI486SXL-G40	3.3-V, 5-V tolerant	40	40, 20†	144-pin QFP‡, and 168-pin PGA§
TI486SXL2-G50	3.3-V, 5-V tolerant	50	25	
TI486SXL-V40	3.3	40	40, 20†	168-pin PGA§
TI486SXL2-V50	3.3	50	25	
TI486SXL-040	5	40	40, 20†	132-pin PGA‡, 144-pin QFP‡, and 168-pin PGA§
TI486SXL2-050¶	5	50	25	

† These microprocessors can be operated as nonclock-doubled 40 MHz or clock-doubled 20/40 MHz.
‡ Pinout and footprint compatible with TI486DLC/E
§ Footprint compatible with 486SX. See Appendix D, *OEM Modifications for 168-Pin CPGA*.
¶ Available in 144-pin ceramic QFP and 168-pin PGA

Figure 1–4. TI486SXL Functional Block Diagram

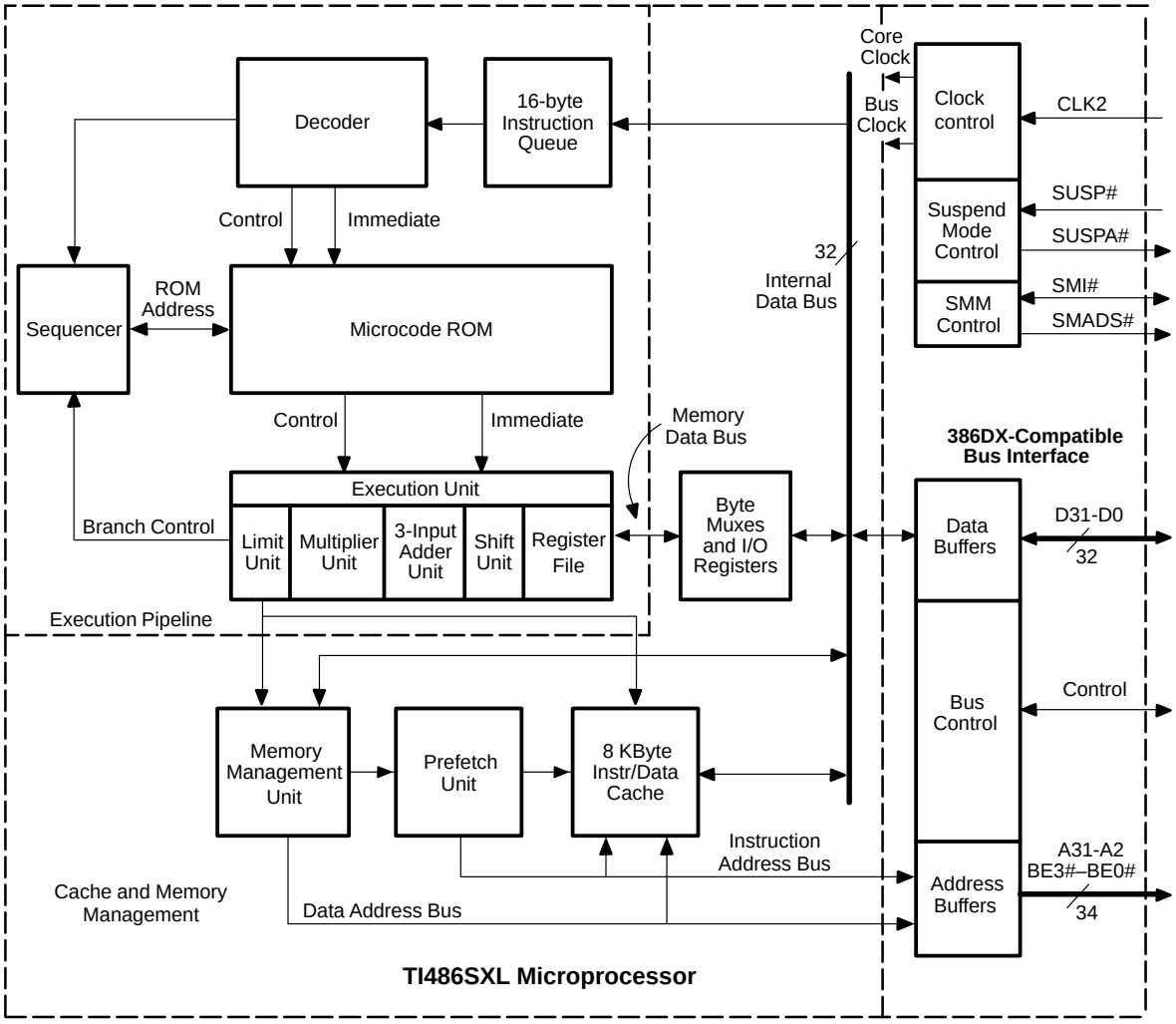
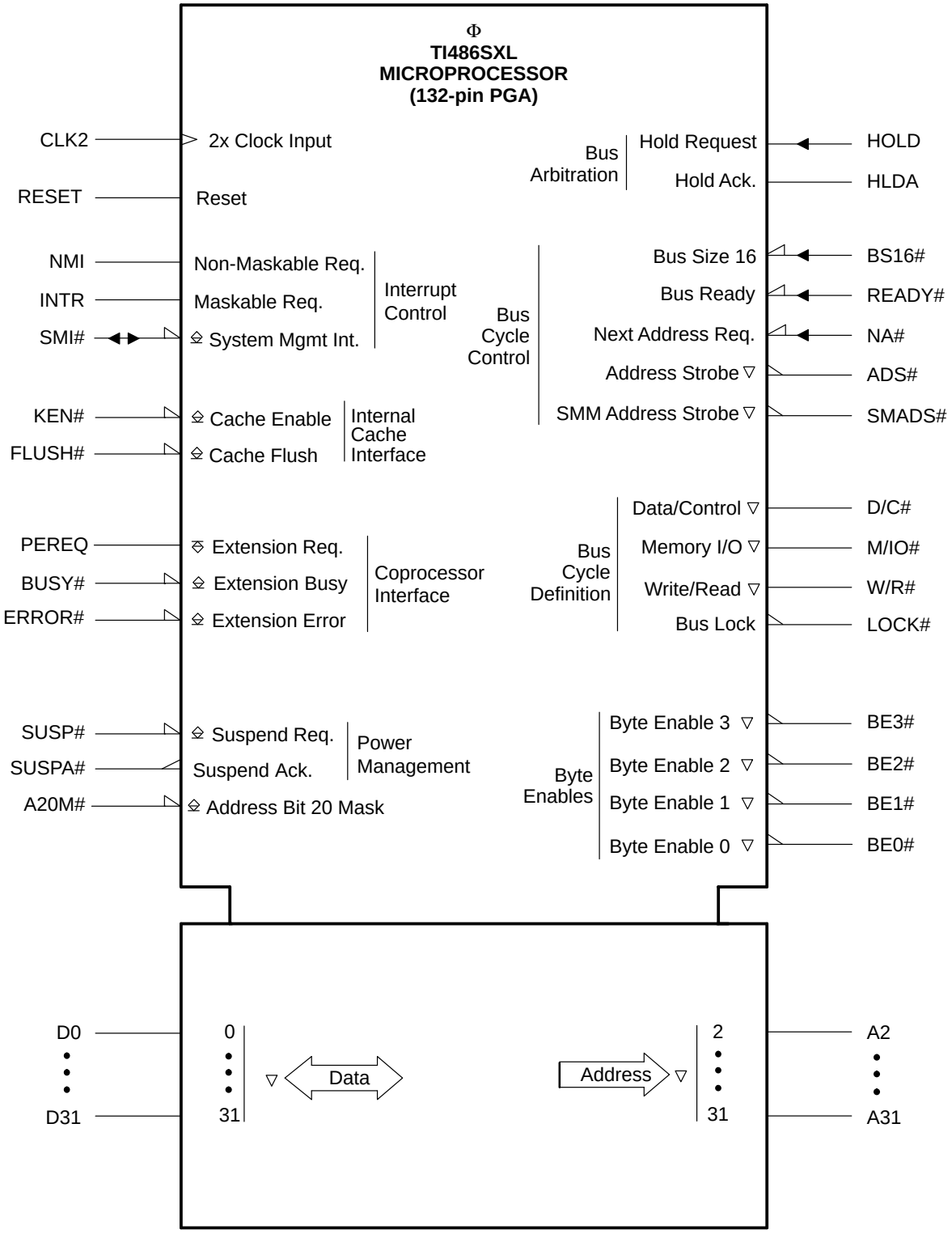
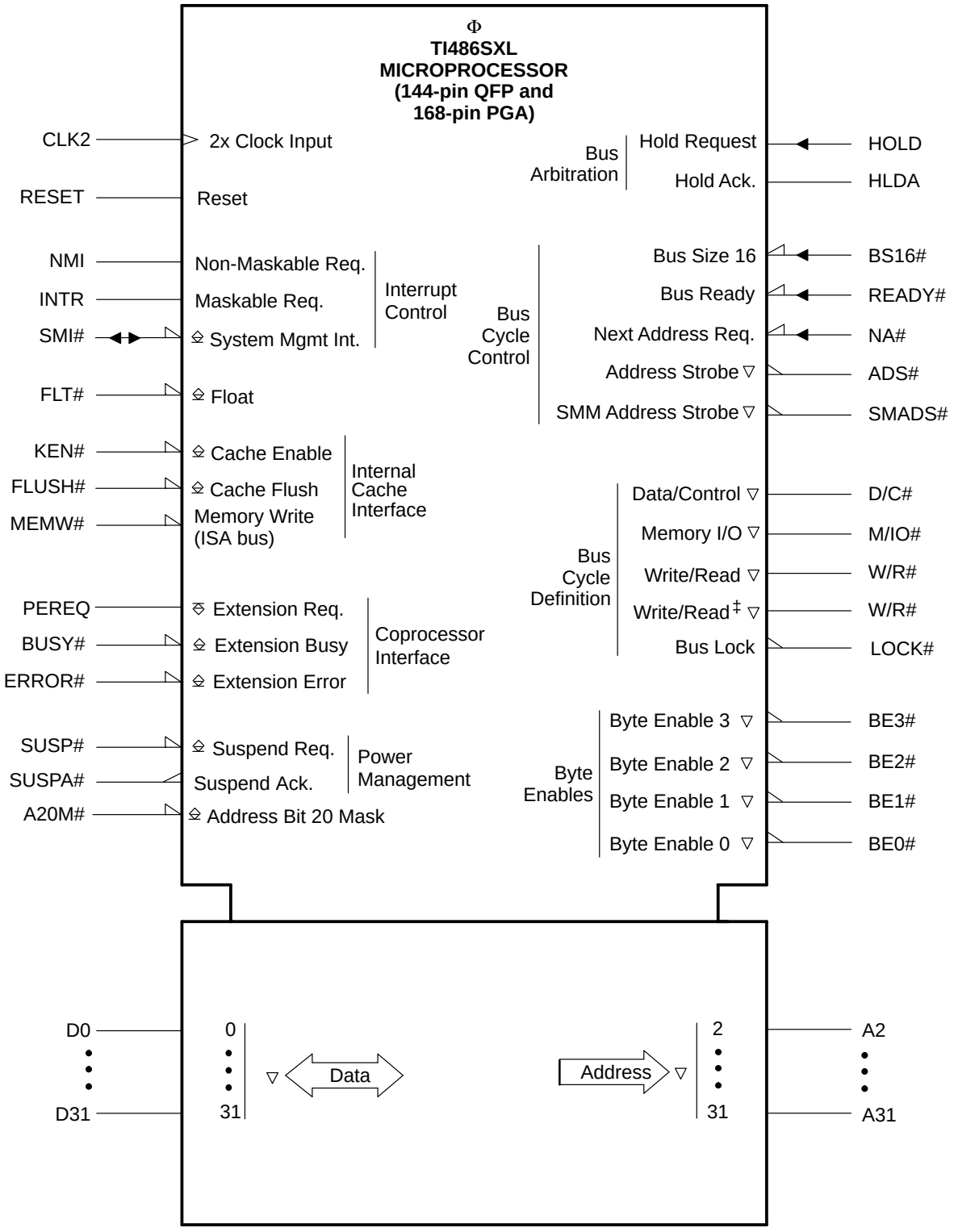


Figure 1–5. TI486SXL Logic Symbol† (132-Pin PGA Package)



† This symbol is in accordance with ANSI/IEEE Std 91-1991 and IEC Publication 617-12.

Figure 1–6. TI486SXL Logic Symbol† (144-Pin QFP and 168-Pin PGA Packages)



† This symbol is in accordance with ANSI/IEEE Std 91-1991 and IEC Publication 617-12.
‡ 144-pin QFP has W/R# on pins 36 and 37. These terminals must be connected together.

The TI486SXL includes two power-management signals (SUSP# and SUSPA#), two cache-interface signals (FLUSH# and KEN#), an A20 mask input (A20M#), and two SMM signals (SMADS# and SMI#) that are additions to the 386DX signal set. The 132-pin PGA TI486SXL has the same signal set as the TI486DLC/E microprocessor while the 144-pin QFP and the 168-pin PGA have two additional inputs, MEMW#, and FLT#. MEMW# is part of the cache interface and FLT# can be used to float the bidirectional and output signals. (See Appendix D, *OEM Modifications for 168-Pin CPGA*.)

The complete list of TI486SXL signals is shown in Figure 1–7 for the 132-pin PGA and Figure 1–8 for the144-pin QFP and 168-pin PGA.

Figure 1–7. TI486SXL Input and Output Signals for 132-Pin PGA Package

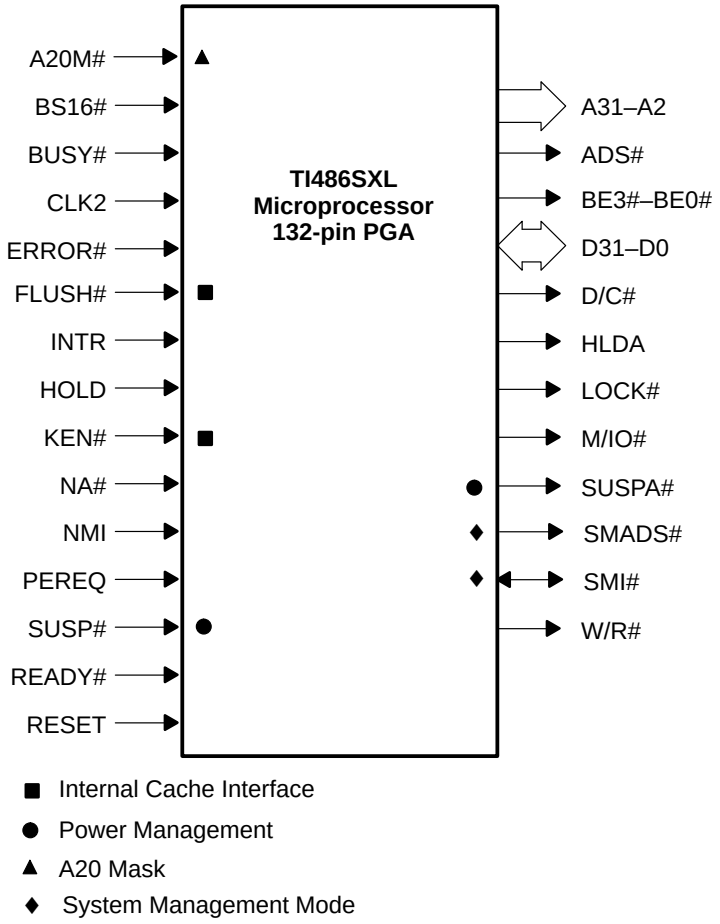
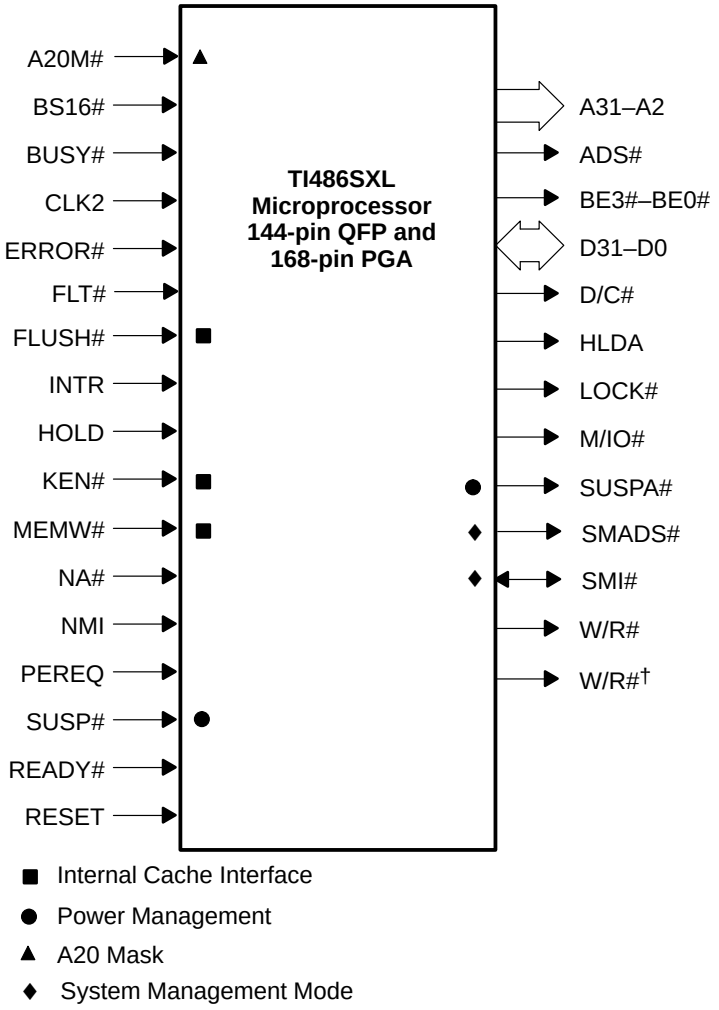


Figure 1–8. TI486SXL Input and Output Signals for 144-Pin QFP and 168-Pin PGA Package



† 144-pin QFP has W/R# on pins 36 and 37. These terminals must be connected together.

1.5 Differences Between the TI486SXLC Series and TI486SXL Series

The TI486SXLC and the 132-pin TI486SXL are the same except for how the pin signals are routed and utilized on the processors. Thus, the bus interfaces are different but the CPU core and the cache/memory management are the same. The TI486SXLC has a physical address range of 16M bytes and the TI486SXL has a physical address range of 4G bytes. Table 1–5 describes the signal differences between the TI486SXLC and TI486SXL.

Table 1–5. TI486SXLC and TI486SXL Signal Differences

Description	TI486SXLC (100-pin QFP)	TI486SXL (132-pin PGA)	TI486SXL (144-pin QFP and 168-pin PGA)
Data bus	16 bits wide (D15–D0)	32 bits wide (D31–D0)	32 bits wide (D31–D0)
Address bus	A23–A1	A31–A2	A31–A2
Byte enables	2 byte enables used (BHE#, BLE#)	4 byte enables used (BE3#–BE0#)	4 byte enables used (BE3#–BE0#)
Float bus signal (FLT#)	supported	not supported	supported
Bus size 16 signal (BS16#)	not supported	supported	supported
MEMW# ISA signal	not supported	not supported	supported

The 144-pin QFP and the 168-pin PGA TI486SXL differ from the TI486SXLC and the 132-pin PGA TI486SXL by the addition of one signal, MEMW#. This signal is part of the cache flush logic that is implemented on-chip in the 144- and 168-pin versions of the TI486SXL. For a more detailed description of this logic, see Appendix C, *Design Considerations and Cache Flush* and Appendix D, *OEM Modifications for 168-Pin CPGA*. The 144-pin QFP and the 168-pin PGA TI486SXL contain the TI486SXLC signal FLT# that is not implemented in the 132-pin PGA TI486SXL. This signal can be used to float all bidirectional and output signals of the TI486SXL microprocessor when it is used in conjunction with an upgrade socket. The 144-pin QFP differs from the 168-pin PGA by the addition of a second W/R# input. As these two W/R# inputs must be connected together, these devices are functionally the same.

1.6 Differences Between the TI486SXL(C) Family and the TI486SLC/DLC Family

The TI486SXLC and the TI486SLC/E are the same in all respects except for the cache size, the cache organization, and the clock-doubled feature. The TI486SXL and the TI486DLC/E are also the same in all respects except for the same new features shown in Table 1–6. Signal differences between the TI486SXLC and the 132-pin PGA TI486SXL, listed in Table 1–5, also apply for the TI486SLC/E and TI486DLC/E, respectively.

Table 1–6. TI486SXL and TI486SLC/DLC Feature Differences

Description	TI486SXL(C) Family	TI486SLC/DLC Family
Cache size	8K bytes	1K byte
Cache organization	Two-way set associative	Two-way set associative or direct mapped
Clock doubled	Supported	Not supported

1.7 Execution Pipeline

The execution path in the TI486SXL(C) family of microprocessors consists of five pipelined stages optimized for minimal instruction-cycle times. These five stages are:

- Code fetch
- Instruction decode
- Microcode ROM access
- Execution
- Memory/register file write-back

These stages have been designed with hardware interlocks that permit execution overlap for successive instructions.

The 16-byte instruction-prefetch queue fetches code in advance and prepares it for decode, helping to minimize overall execution time. The instruction decoder then decodes four bytes of instructions per clock, eliminating the need for a queue of decoded instructions. Sequential instructions are decoded quickly and provided to the microcode. Nonsequential operations do not have to wait for a queue of decoded instructions to be flushed and refilled before execution continues. As a result, both sequential and nonsequential instruction execution times are minimized.

The execution stage takes advantage of a RISC-like, single-cycle execution unit and a 16-bit hardware multiplier. The write-back stage provides single-cycle, 32-bit access to the on-chip cache and posts all writes to the cache and system bus using a two-deep write buffer. Posted writes allow the execution unit to proceed with program execution while the bus-interface unit completes the write cycle.

1.8 On-Chip Cache

The 8K-byte, 32-bit on-chip cache in the TI486SXL(C) family of microprocessors maximizes overall performance by quickly supplying instructions and data to the internal execution pipeline. An external memory access takes a minimum of two clock cycles (zero wait states). For cache hits, the TI486SXL series eliminates these two clock cycles by overlapping cache accesses with normal execution pipeline activity. In addition, bus bandwidth is gained by presenting instructions and data to the execution pipeline at up to 32 bits at a time compared to 16 bits per cycle for an external memory access.

The TI486SXL(C) cache is an 8K-byte, write-through unified instruction and data cache with lines that are allocated only during memory read cycles. The cache is configured as two-way set associative, and the cache organization consists of 1024 sets each containing two lines of four bytes each.

1.9 Clock-Doubled Mode

The TI486SXL(C) family of microprocessors is designed with a clock-doubled feature that provides an immediate performance increase and upgrade path from the TI486SLC/DLC family of products. The clock-doubled feature can be enabled using software by setting bit 6 of the Configuration Control register 0.

When the microprocessor is in clock-doubled mode, the internal core is operating at the CLK2 frequency while the external bus interface remains at half the CLK2 frequency. This provides a speed increase in the on-chip cache, the instruction decode, and the instruction execution while the external interface remains the same.

In addition to the clock-doubled feature, the TI486SXL(C) microprocessor family also supports dynamic clock scaling that enables the CLK2 input to be scaled up or down. To take advantage of this feature (scaling or stopping the CLK2 input), the processor must first be brought into the nonclock-doubled mode. Dynamic clock scaling is transparent to the user since the processor continues instruction execution in nonclock-doubled mode until the desired frequency is reached within the PLL lock range to initiate clock-doubled mode. This allows for increased bandwidth on demand without restriction to the user.

1.10 Power Management

The TI486SXL(C) family incorporates advanced power-management features such as suspend mode, static operation, and operation at 3.3 V. These capabilities are attractive for battery-powered notebook and energy-efficient desktop PC systems.

1.10.1 System-Management Mode (SMM)

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#) that has a higher priority than any other interrupt. While running in protected SMM address space, the SMI interrupt routine can execute without interfering with the operating system or application programs.

After receiving 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. The location and size of the SMM memory is programmable in the TI486SXL(C) microprocessor family. Seven SMM instructions have been added to the 486 instruction set that permit saving and restoring the total CPU state when in SMM mode.

1.10.2 Suspend Mode and Static Operation

The power-management features in the TI486SXL(C) family of microprocessors allow a dramatic reduction in the current required when the microproces-

sor is in suspend mode (typically using less than three percent of the operating current). Suspend mode is entered by either a hardware- or a software-initiated action. Using hardware to initiate suspend mode involves a two-pin handshake using the SUSP# and SUSPA# signals.

The software initiates suspend mode through execution of the HALT instruction. Once in suspend mode, the microprocessor power consumption can be further reduced by stopping the external clock input.

Note:

For the clock-doubled versions of the TI486SXL(C) microprocessor family, suspend mode can be initiated while in clock-doubled mode as long as the external input clock is not stopped. The external input clock can be stopped after the microprocessor has been put into nonclock-doubled mode.

Since these microprocessors are static devices, no internal CPU data is lost when the clock input is stopped.

1.10.3 3.3-V Operation

The TI486SXLC-V and TI486SXLC2-V versions operate from a 3.3-V supply. Power consumed at 3.3 V is typically only 30 percent of the power consumed while operating at 5 V. The TI486SXLC-V25 operates at 25-MHz.

The TI486SXL-V and TI486SXL2-V versions also operate from a 3.3-V supply. Again, power consumed at 3.3 V is typically only 30 percent of the power consumed by a microprocessor operating at 5 V. The TI486SXL-V40 can be operated in clock-doubled mode at 40-MHz core and 20-MHz bus speeds, or it can be operated in nonclock-doubled mode with both the core and bus speeds at 40 MHz. The TI486SXL2-V50 operates at 50 MHz core and 25-MHz bus speeds in the clock-doubled mode.

1.10.4 Mixed 3.3-V and 5-V Operation

The TI486SXL-G and TI486SXL2-G versions operate from both a 3.3-V and a 5-V supply. These microprocessors feature 5-V tolerant inputs and outputs, which means that they can be incorporated in system designs that utilize both 3.3-V and 5-V devices. These devices can be used in 3.3-V-only systems by connecting the 5-V supply pin (V_{CC5}) to the 3.3-V supply. The microprocessor power consumption is typically only 30 percent of the power consumed by a microprocessor operating at 5 V. The TI486SXL-G40 can be operated in clock-doubled mode at 40-MHz core and 20-MHz bus speeds, or it can be operated in nonclock-doubled mode with both the core and bus speeds at 40 MHz. The TI486SXL2-G50 operates at 50-MHz core and 25-MHz bus speeds in the clock-doubled mode.

Chapter 2

Programming Interface

In this chapter, the internal operations of the TI486SXL(C) family of microprocessors are described mainly from an application programmer's point of view. Included in this chapter are descriptions of processor initialization, the register sets, memory addressing, various types of interrupts, system-management mode, and the shutdown and halt process. Overviews of real, virtual-8086, and protected operating modes are also included.

Topic **Page**

2.1 Processor Initialization

Each TI486SXL(C) family microprocessor is initialized when the RESET signal is asserted. The processor is placed in real mode and the registers listed in Table 2–1 or Table 2–2 are set to their initialized values. RESET invalidates and disables the cache and turns off paging. For the clock-doubled versions of the TI486SXL(C) microprocessor family, RESET returns the processor to the nonclock-doubled mode. When RESET is asserted, the microprocessor terminates all local bus activity and all internal execution. During the time that RESET is asserted, the internal pipeline is flushed and no instruction execution or bus activity occurs.

Approximately 350 to 450 CLK2 clock cycles (additional 2^{20} + 60 if self-test is requested) after negation of RESET, the processor begins executing instructions at the top of physical memory (address location FF FFF0h for the TI486SXLC series and FFFF FFF0h for the TI486SXL series). When the first intersegment JUMP or CALL is executed, address lines A23–A20 for the TI486SXLC series or A31–A20 for the TI486SXL series are driven low for code-segment-relative memory-access cycles. While these address lines are low, the microprocessor executes instructions only in the lowest 1M byte of physical address space until system-specific initialization occurs via program execution.

Table 2–1. TI486SXLC Initialized Register Contents

Register	Register Name	Initialized Contents	Comments
EAX	Accumulator	xx xxxxh	00 0000h indicates self-test passed.
EBX	Base	xx xxxxh	
ECX	Count	xx xxxxh	
EDX	Data	xx 0400h + Revision ID	Revision ID = 10h
EBP	Base Pointer	xx xxxxh	
ESI	Source Index	xx xxxxh	
EDI	Destination Index	xx xxxxh	
ESP	Stack Pointer	xx xxxxh	
EFLAGS	Flag Word	00 0002h	
EIP	Instruction Pointer	00 FFF0h	
ES	Extra Segment	0000h	Base address set to 00 0000h Limit set to FFFFh
CS	Code Segment	F000h	Base address set to 00 0000h Limit set to FFFFh
SS	Stack Segment	0000h	
DS	Data Segment	0000h	Base address set to 00 0000h Limit set to FFFFh
FS	Extra Segment	0000h	
GS	Extra Segment	0000h	
IDTR	Interrupt-Descriptor Table	Base=0, Limit=3FFh	
CR0	Machine Status Word	00 0010h	
CCR0	Configuration Control 0	00h	
CCR1	Configuration Control 1	xx xxx0 (binary)	
ARR1	Address Region 1	000Fh	4G-byte noncacheable region
ARR2	Address Region 2	0000h	
ARR3	Address Region 3	0000h	
ARR4	Address Region 4	0000h	
DR7	Debug	00 0000h	

Note: x = Undefined value

Table 2–2. TI486SXL Initialized Register Contents

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 0421h + Revision ID	Revision ID = 10h
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 0000 0000h Limit set to FFFFh
SS	Stack Segment	0000h	
DS	Data Segment	0000h	Base address set to 0000 0000h Limit set to FFFFh
FS	Extra Segment	0000h	
GS	Extra Segment	0000h	
IDTR	Interrupt-Descriptor Table	Base=0, Limit=3FFh	
CR0	Machine Status Word	0000 0010h	
CCR0	Configuration Control 0	00h	
CCR1	Configuration Control 1	xxxx xxx0 (binary)	
ARR1	Address Region 1	000Fh	4G-byte noncacheable region
ARR2	Address Region 2	0000h	
ARR3	Address Region 3	0000h	
ARR4	Address Region 4	0000h	
DR7	Debug	0000 0000h	

Note: x = Undefined value

2.2 Real Mode Versus Protected Mode

When powered up or reset, the microprocessor is initialized to real mode. Real mode establishes conditions that are backward compatible with the 8086/8088 microprocessors. Addressing capabilities are limited to the range that is available on those two microprocessors, and the default operand size is 16 bits.

The microprocessor can be switched from the real mode into protected mode, where the extended capabilities of the TI486SXL(C) are available for use. Protected mode provides enhanced memory management capabilities that include segment- and page-level protection.

Table 2–3 provides a comparison of real mode and protected mode. The microprocessor is in protected mode when the PE bit in Control register 0 is set. After this bit is set, an intersegment JMP is used to load the CS register and to flush the instruction-decode pipeline.

Table 2–3. Comparison of Real Mode and Protected Mode

Feature	Real Mode	Protected Mode
Physical memory	Limited to 1M byte.	Limited to 4G bytes. Virtual memory of up to 4T bytes is available.
Default operand size	Normally 16 bits.	Can be 16 or 32 bits.
Segment size	Fixed at 64K bytes.	Variable from 1 byte to 4G bytes.
Physical address	Generated by multiplying the segment register value by 16 and adding an offset to the product.	Generated by applying paging, if enabled, to linear addresses. Linear addresses are generated by adding an offset to a value calculated from information contained in segment descriptors. The value in a segment register determines which of several possible segment descriptors is used.
Segment access	Attempted access beyond the end of a segment is monitored.	Segments can be given combinations of read, write, and execute permissions. Attempted access beyond the end of a segment is monitored.
Code privileged	No privileged code.	Code can have one of four privilege levels, with some processor instructions restricted to the most privileged level.

2.3 Instruction-Set Overview

The TI486SXL(C) microprocessor family instruction set can be divided into eight types of operations:

- Arithmetic
- Bit manipulation
- Control transfer
- Data transfer
- High-level-language support
- Operating-system support
- Shift/rotate
- String manipulation

All instructions operate on as few as zero operands and as many as three operands. An NOP (no operation) instruction is an example of a zero-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 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. 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 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 7, *Instruction Set*, lists each instruction in the TI486SXL(C) microprocessor family instruction set along with the associated opcodes, execution clock counts, and effects on the Flag Word register.

2.3.1 Lock Prefix

The LOCK prefix can be placed before certain instructions that read, modify, and 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).

2.3.2 Register Sets

The TI486SXL(C) microprocessor has 43 accessible registers grouped into two sets. The application register set contains the registers frequently used by applications programmers. The system register set contains the registers typically reserved for use by operating-systems programmers.

The application register set is made up of:

- Eight 32-bit General Purpose registers
- Six 16-bit Segment registers
- One 32-bit Flag Word register
- One 32-bit Instruction Pointer register

The system register set is made up of the remaining registers that include:

- Three 32-bit Control registers
- Two 48-bit and two 16-bit System Address registers
- Two 8-bit and four 16-bit (TI486SXLC) or 24-bit (TI486SXL) Configuration registers
- Six 32-bit Debug registers
- Five 32-bit Test registers

Each application register is discussed in Section 2.4, *Application Register Set*, page 2-10.

Each system register is discussed in Section 2.5, *System Register Set*, page 2-16.

2.3.3 Address Spaces

The microprocessor can directly address either memory or I/O space. Figure 2–1 and Figure 2–2 illustrate the range of addresses available for memory address space and I/O address space.

Figure 2–1. TI486SXLC Memory and I/O Address Spaces

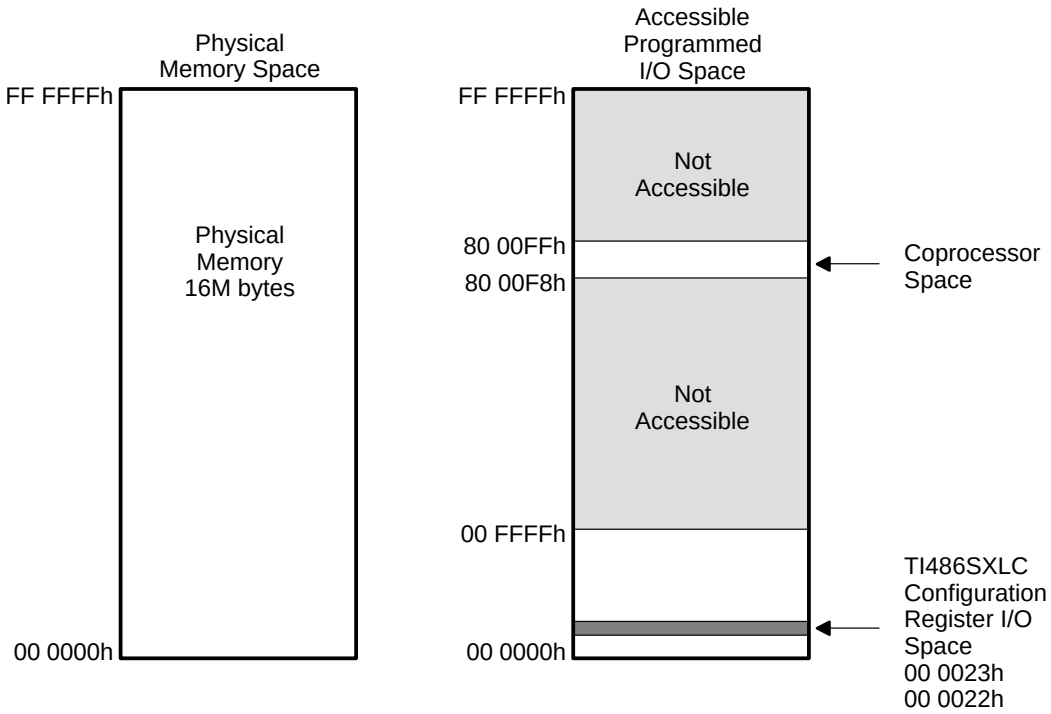
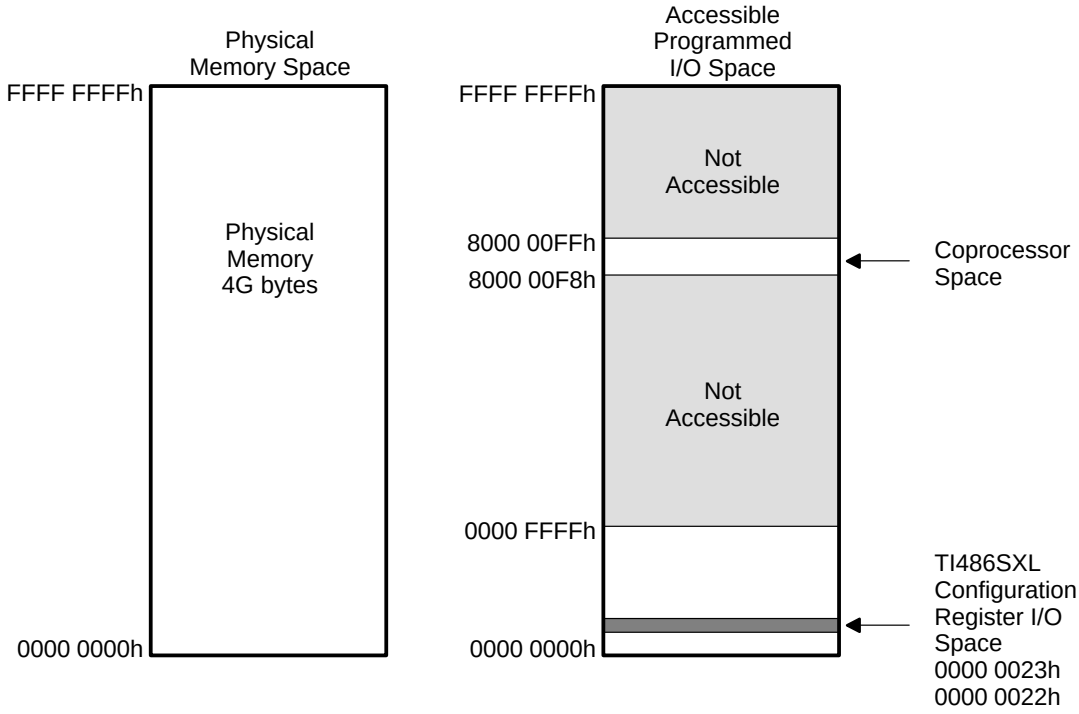


Figure 2–2. TI486SXL Memory and I/O Address Spaces



2.3.3.1 Memory Address Space Range

For the TI486SXLC series, the addresses for physical memory range between 00 0000h and FF FFFFh (16M bytes). For the TI486SXL series, the addresses for physical memory range between 0000 0000h and FFFF FFFFh (4G bytes). 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.

Section 2.6, *Memory Address Space*, page 2-37, discusses in detail:

- Memory addressing modes that are used to calculate the physical address
- Memory management mechanisms, segmentation, and paging that can be used to protect address spaces and create an environment that lets a small amount of physical memory simulate a large address space.

2.3.3.2 I/O Address Space Range

The accessible I/O address space for both the TI486SXLC and TI486SXL microprocessors ranges between 00 0000h and 00 FFFFh (64K bytes). The coprocessor communication space for the TI486SXLC series exists in upper I/O space between 80 00F8h and 80 00FFh. The coprocessor communication space for the TI486SXL series exists in the upper I/O space between 8000 00F8h and 8000 00FFh. These coprocessor I/O ports are automatically accessed by the CPU whenever an ESC opcode (IN or OUT I/O instruction) is executed. The I/O locations 22h and 23h are used for Configuration register access on all versions of the TI486SXL(C) microprocessors.

The I/O address space is accessed using IN and OUT instructions pointing to addresses that are referred to as ports. The accessible I/O address space is 64K bytes and can be accessed as 8-bit, 16-bit, or 32-bit ports. The execution of any IN or OUT instruction causes M/IO# to be driven low, thereby selecting the I/O space instead of memory space for loading or storing data. The upper eight address bits of the TI486SXLC processor and the upper sixteen bits of the TI486SXL processor are driven low during IN and OUT instruction port accesses.

The microprocessor 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 microprocessor index range (C0h to CFh) result in external I/O cycles and do not affect the on-chip Configuration registers. Reads of port 22h are always directed off-chip.

2.4 Application Register Set

The Application register set (Figure 2–3) 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.

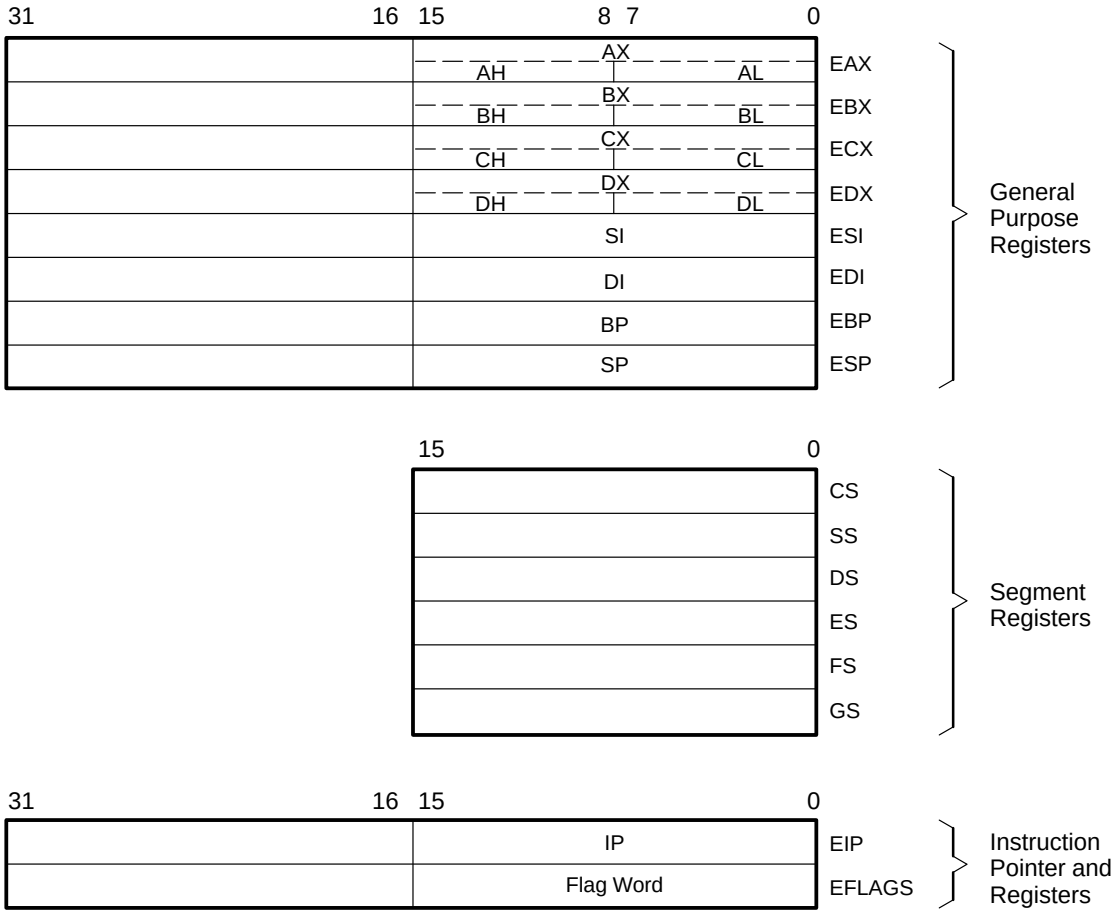
The contents of General Purpose registers are frequently modified by assembly language instructions and typically contain arithmetic and logical-instruction operands.

The Segment registers contain segment selectors that index into tables located in memory. These tables hold the base address for each segment and other information related to memory addressing.

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

The Instruction Pointer is a 32-bit register that points to the next instruction that the processor executes. This register is automatically incremented by the processor as execution progresses.

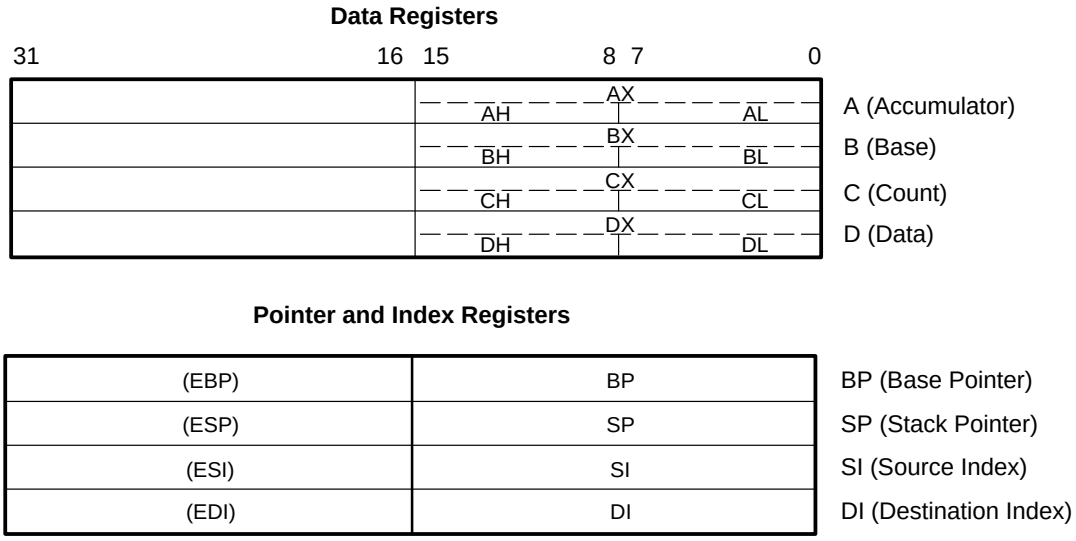
Figure 2–3. Application Register Set



2.4.1 General Purpose Registers

The General Purpose registers are divided into four Data, two Pointer, and two Index registers as shown in Figure 2–4.

Figure 2–4. General Purpose Registers



2.4.1.1 Data Registers

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 source operand value specified by an instruction is smaller than the specified destination register, the upper bytes of the destination register are not affected when the operand is written to the register.

2.4.1.2 Pointer and Index Registers

The Pointer and Index registers are:

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

These registers can be addressed as 16- or 32-bit registers, with the E prefix indicating 32 bits. These registers can be used as General Purpose registers; however, some instructions use a fixed assignment of these registers. For example, the 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 double-precision multiply and divide, I/O access, string operations, translate, loop, variable shift and rotate, and stack operations.

The TI486SXL(C) processors implement 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. The EBP register can be used to reference data passed onto the stack during procedure calls. Local data can also be placed on the stack and referenced relative to BP. This register provides a mechanism to access stack data in high-level languages.

2.4.2 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 memory locations of code, data, and stack references.

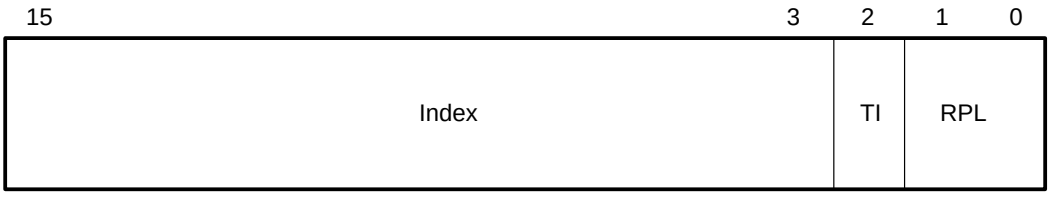
There are six 16-bit Segment registers:

CS	Code Segment
DS	Data Segment
FS	Additional Data Segment
GS	Additional Data Segment
SS	Stack Segment
ES	Extra 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 2–5.

Figure 2–5. Segment Selector Register



TI = Table Indicator

RPL = Requested Privilege Level

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 TI bit of the selector defines the descriptor table into which the index points. 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.

The requested privilege level (RPL) field contains a 2-bit segment privilege level (00 = most privileged, 11 = 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 program, the RPL overrides the current privilege level (CPL) and the EPL is the lower privilege level. If the RPL bits indicate more privilege than the program, the current privilege level overrides the RPL and again the EPL is the lower privilege level.

When a Segment register is loaded with a segment selector, the segment base, the segment limit, and the 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.

The processor automatically selects a default Segment register for memory references. Table 2–4 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 can 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 ES, FS, and GS Segment registers.

Table 2–4. 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, DS, ES, FS, GS

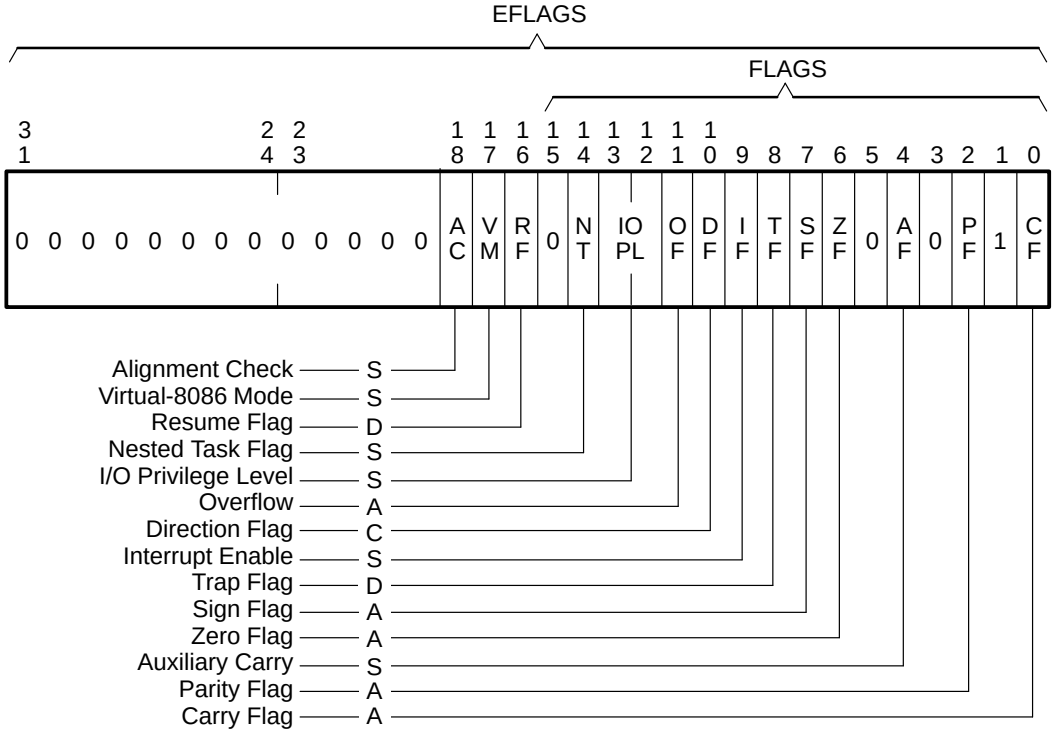
2.4.3 Instruction Pointer Register

The (extended) Instruction Pointer (EIP) register shown in Figure 2–3 on page 2-10 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).

2.4.4 Flag Word Register

The Flag Word register, EFLAGS, contains status information and controls certain operations on the microprocessor. The lower 16 bits of this register are referred to as the Flag register, FLAGS, that is used when executing 8086 or 80286 code. The flag bits are shown in Figure 2–6 and defined in Table 2–5.

Figure 2–6. EFLAGS Register



Note: A = arithmetic flag, D = debug flag, S = system flag, C = control flag
0 or 1 indicates reserved

Table 2–5. EFLAGS Definitions

Bit Position	Name	Function
0	CF	Carry flag. CF is set when an operation results in a carry out of (addition) or borrow into (subtraction) the most significant bit; otherwise, CF is cleared.
2	PF	Parity flag. PF is set when the low-order eight bits of the result contain an even number of ones; otherwise, PF is cleared.
4	AF	Auxiliary carry flag. AF is set when an operation results in a carry out of (addition) or borrow into (subtraction) bit position 3; otherwise, AF is cleared.
6	ZF	Zero flag. ZF is set if result is zero; otherwise, ZF is cleared.
7	SF	Sign flag. SF is set equal to high-order bit of result (0 indicates positive, 1 indicates negative).
8	TF	Trap enable flag. Once TF is 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 IF is 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.
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. RF is 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 flag. If VM is 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 flag 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.

2.5 System Register Set

The System register set (Figure 2–7) consists of registers typically used by system-level programmers who generate operating systems and memory-management programs.

The Control registers control functions of the microprocessor such as paging, coprocessor functions, and segment protection. When paging is enabled and a paging exception occurs, the Control registers retain the linear address of the access that caused the exception.

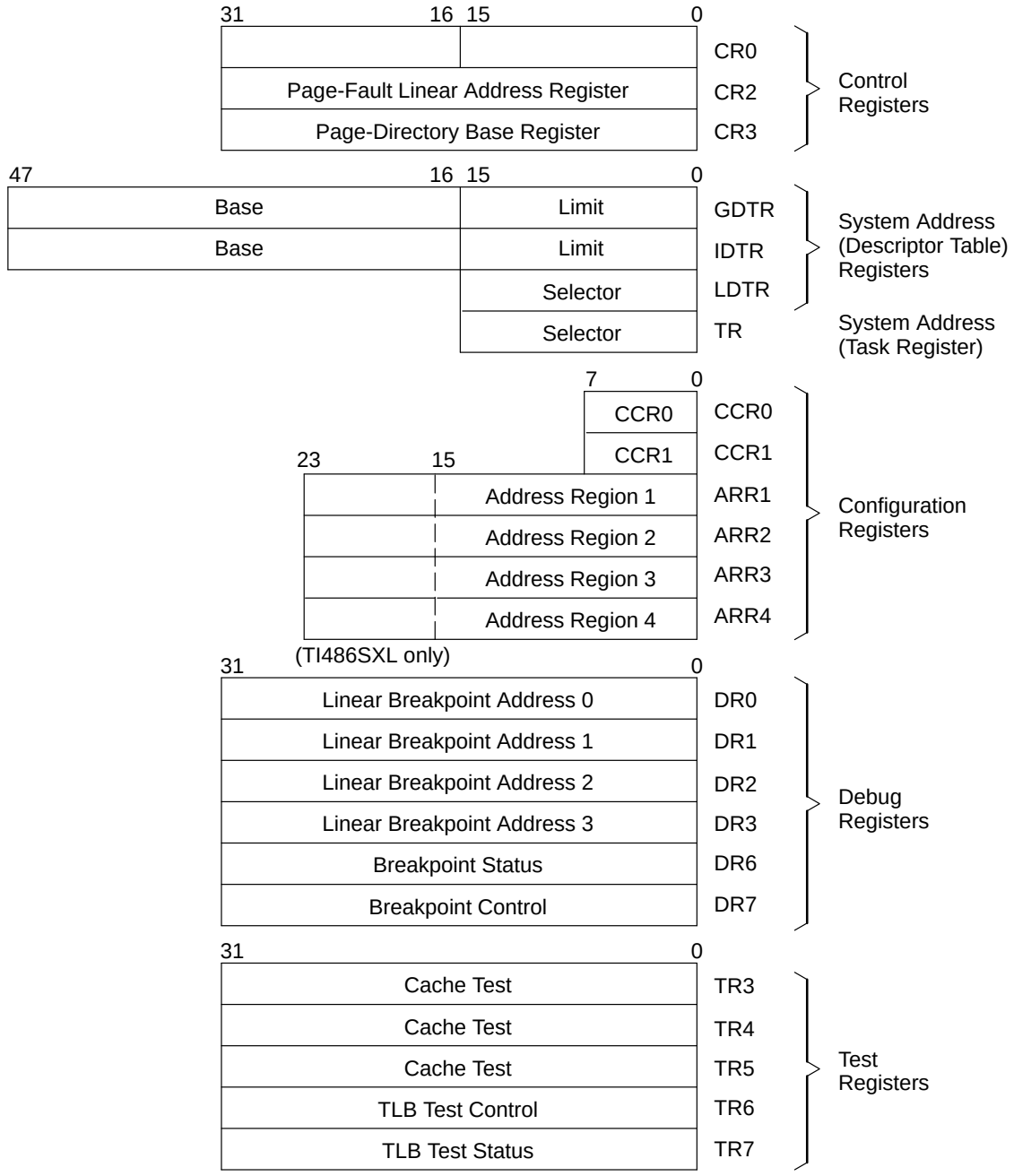
The Descriptor Table registers and the Task register are referred to as System Address or Memory Management registers. These registers consist of two 48-bit and two 16-bit registers. These registers specify the location of the data structures that control the segmentation used by the microprocessor. Segmentation is a method of memory management.

The Configuration registers control the clock-doubling operation (for the TI486SXLC2 and TI486SXL2), on-chip cache operation, power-management features, and system-management mode. The clock-doubling, cache, power-management, and SMM features can be enabled or disabled by writing to these registers. Noncacheable areas of physical memory are also defined through these registers.

The Debug registers provide debugging facilities for the 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 8K-byte cache and the translation lookaside buffer (TLB). The TLB is used as a cache for translating linear addresses to physical addresses when paging is enabled. In the following sections, the System register set is described in greater detail.

Figure 2–7. System Register Set

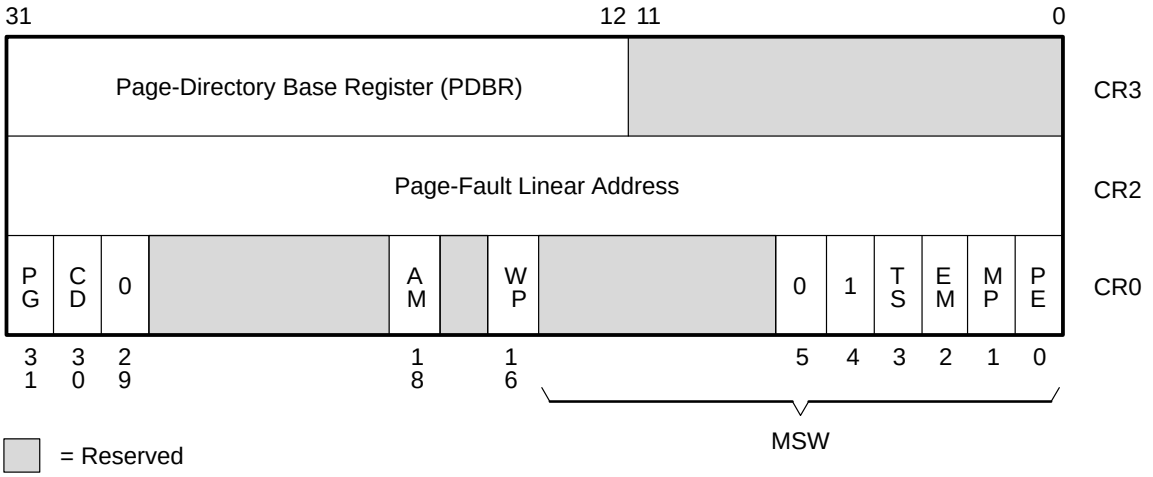


CCR0 = Configuration Control 0
CCR1 = Configuration Control 1

2.5.1 Control Registers

The Control registers (CR0, CR2, and CR3) are shown in Figure 2–8. The CR0 register contains system control flags that 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 2–6. The reserved bits in CR0 should not be modified.

Figure 2–8. Control Registers



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. CR3 contains the 20-bit base address of the page directory. The page directory must always be aligned to a 4K-byte page boundary; therefore, the lower 12 bits of CR3 are reserved.

When the microprocessor operates in protected mode, any program can read the Control registers. Privilege level 0 (most privileged) programs can modify the contents of these registers.

Table 2–6. CR0 Bit Definitions

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 (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 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 1 during normal operations.
2	EM	Emulate processor extension. If EM = 1, all floating-point instructions cause a 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 device-not-available (DNA) fault. If MP = 1 and TS = 1, a WAIT instruction also causes a DNA fault.
4	1	Reserved. Do not modify.
5	0	Reserved. Do not modify.
16	WP	Write protect. Protects read-only pages from supervisor write access. The 386-type CPU allows a read-only page to be written from privilege levels 0–2. The TI486SXL(C) CPU is compatible with the 386-type CPU when WP = 0. 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.
29	0	Reserved. Do not modify.
30	CD	Cache disable. If CD = 1, cache is no longer filled. 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. If PG = 1 and protected mode is enabled (PE = 1), paging is enabled.

2.5.2 Descriptor-Table Registers and Descriptors

The Global-, Interrupt-, and Local-Descriptor-Table registers (GDTR, IDTR and LDTR) specify the location of the data structures that control segmented memory management.

2.5.2.1 Descriptor-Table (System-Address) Registers

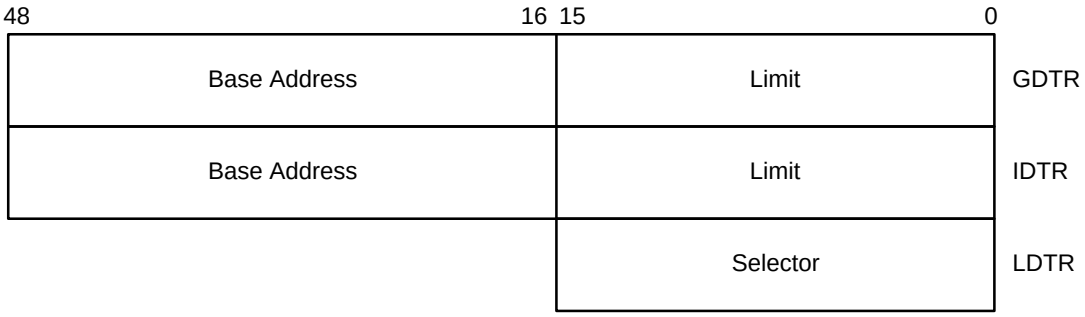
The GDTR, IDTR, and LDTR, shown in Figure 2–9, are loaded using the LGDT, LIDT, and LLDT instructions, respectively. The values of these registers are stored using the corresponding store instructions. The GDTR and IDTR load instructions are privileged instructions when operating in protected mode. The LDTR can be accessed only in protected mode.

The Global-Descriptor-Table register (GDTR) holds a 32-bit base address and a 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 microprocessor 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 base address and a 16-bit limit for the interrupt-descriptor table (IDT). The IDT is an array of 256 8-byte interrupt descriptors, each of which points to an interrupt service routine. Every interrupt that can occur in the system must have an associated entry in the IDT. The contents of the IDTR are completely visible to the programmer.

Figure 2–9. Descriptor-Table (System-Address) Registers



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 not access the GDT again until the LDTR is reloaded. If the LDT description 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 for isolating a given task's segments from other tasks in the system.

2.5.2.2 Descriptors

The three types of descriptors are:

- Application-segment descriptors that define code, data, and stack segments
- System-segment descriptors that define an LDT segment or a task-state segment (TSS)
- 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 be located only in the GDT. Depending on the gate type, gate descriptors can be located in the GDT, LDT, or IDT. Figure 2–10 illustrates the descriptor format for both application-segment descriptors and system-segment descriptors. Table 2–7 lists the corresponding bit definitions.

Figure 2–10. Application- and System-Segment Descriptors

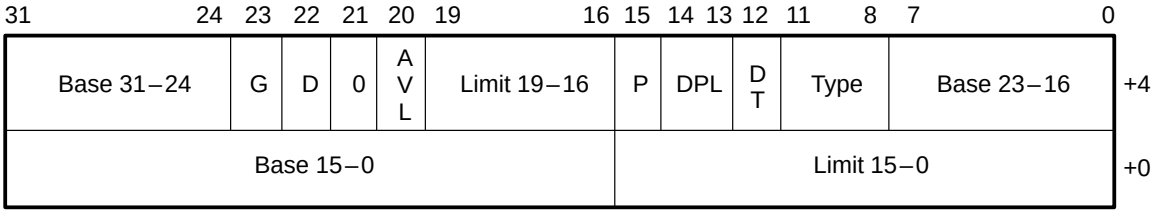


Table 2–7. Segment Descriptor Bit Definitions

Bit Position	Memory Offset	Name	Description
31–24 7–0 31–16	+4 +4 +0	Base 31–24 Base 23–16 Base15–0	Segment base address. A 32-bit linear address that points to the beginning of the segment.
19–16 15–0	+4 +0	Limit 19–16 Limit 15–0	Segment limit. In real mode, segment limit is always 64K bytes (0FFFFh).
23	+4	G	Limit granularity: 0 = byte granularity 1 = 4K-byte (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	+4	E	Application descriptor (DT = 1): 0 = data 1 = executable
10	+4	C/D	If E is 0: 0 = expand up, limit is upper bound of segment 1 = expand down, limit is lower bound of segment If E is 1: 0 = nonconforming 1 = conforming (runs at privilege level of calling procedure)
9	+4	R/W	If E is 0: 0 = nonreadable 1 = readable If E is 1: 0 = nonwritable 1 = writable
8	+4	A	0 = not accessed 1 = accessed

Gate descriptors provide protection for executable segments operating at different privilege levels. Figure 2–11 illustrates the format for gate descriptors and Table 2–8 lists the corresponding bit definitions.

Task-gate descriptors are used to switch the CPU's context during a task switch. The selector portion of the task-gate descriptor locates a task-state segment. Task-gate descriptors can be located in the GDT, LDT, or IDT.

Figure 2–11. Gate Descriptor

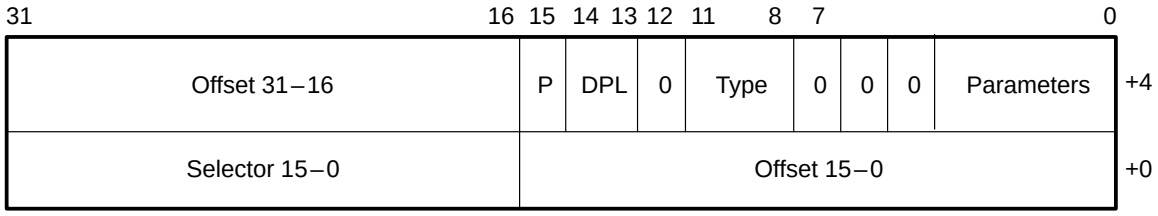


Table 2–8. Gate Descriptor Bit Definitions

Bit Position	Memory Offset	Name	Description
31–16 15–0	+4 +0	Offset 31–16 Offset 15–0	Offset used during a call gate to calculate the branch target
31–16	+0	Selector 15–0	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 gate 0101 = tack gate 0110 = 16-bit interrupt gate 0111 = 16-bit trap gate 1100 = 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

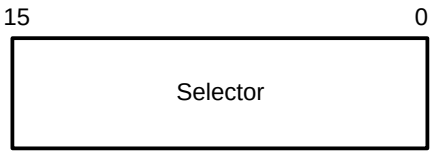
Interrupt-gate descriptors are used to enter a hardware interrupt service routine. Trap-gate descriptors are used to enter exceptions or software interrupt service routines. Trap-gate and interrupt-gate descriptors can be located only in the IDT.

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.

2.5.3 Task Register

The Task register (TR) holds a 16-bit selector for the current TSS table as shown in Figure 2–12. The TR is loaded and stored via the LTR and STR instructions, respectively. The TR can be accessed only during protected mode and can be loaded only when the privilege level is 0 (most privileged).

Figure 2–12. Task (System-Address) Register



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 TSS (32-bit) or a 286-type TSS (16-bit) as shown in Figure 2–13 and Figure 2–14. An I/O permission bit map is referenced in the 32-bit TSS by the I/O map base address.

Figure 2–13. 32-Bit Task-State Segment (TSS) Table

31 – 16	15 – 0	
I/O Map Base Address	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 T	+64h
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Selector For Task's LDT	+60h
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	GS	+5Ch
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	FS	+58h
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	DS	+54h
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS	+50h
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	CS	+4Ch
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	ES	+48h
EDI		+44h
ESI		+40h
EBP		+3Ch
ESP		+38h
EBX		+34h
EDX		+30h
ECX		+2Ch
EAX		+28h
EFLAGS		+24h
EIP		+20h
CR3		+1Ch
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS for CPL = 2	+18h
ESP for CPL = 2		+14h
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS for CPL = 1	+10h
ESP for CPL = 1		+Ch
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS for CPL = 0	+8h
ESP for CPL = 0		+4h
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Back Link (Old TSS Selector)	+0h

0 = Reserved

Figure 2–14. 16-Bit Task-State Segment (TSS) Table

Selector For Task's LDT	+2Ah
DS	+28h
SS	+26h
CS	+24h
ES	+22h
DI	+20h
SI	+1Eh
BP	+1Ch
SP	+1Ah
BX	+18h
DX	+16h
CX	+14h
AX	+12h
FLAGS	+10h
IP	+Eh
SP For Privilege Level 2	+Ch
SS For Privilege Level 2	+Ah
SP For Privilege Level1	+8h
SS For Privilege Level 1	+6h
SP For Privilege Level 0	+4h
SS For Privilege Level 0	+2h
Back Link (Old TSS Selector)	+0h

2.5.4 Configuration Registers

The TI486SXL(C) family microprocessors contain six registers that do not exist on other 80x86 microprocessors. These registers include two Configuration Control registers (CCR0 and CCR1) and four Address Region registers (ARR1 through ARR4) as listed in Table 2–9 and Table 2–10. The CCR and ARR registers exist in I/O memory space and are selected by a register index number via I/O port 22h. I/O port 23h is used for data transfer.

Table 2–9. TI486SXLC Configuration Control Registers

Register Name	Register Index	Width
Configuration Control 0 (CCR0)	C0h	8
Configuration Control 1 (CCR1)	C1h	8
Address Region 1 (ARR1)	C5h–C6h	16
Address Region 2 (ARR2)	C8h–C9h	16
Address Region 3 (ARR3)	CBh–CCh	16
Address Region 4 (ARR4)	CEh–CFh	16

Note: The following register index numbers are reserved: C2h, C3h, C4h, C7h, CAh, CDh, and D0h through FFh.

Table 2–10. TI486SXL Configuration Control Registers

Register Name	Register Index	Width
Configuration Control 0 (CCR0)	C0h	8
Configuration Control 1 (CCR1)	C1h	8
Address Region 1 (ARR1)	C4h–C6h	24
Address Region 2 (ARR2)	C7h–C9h	24
Address Region 3 (ARR3)	CAh–CCh	24
Address Region 4 (ARR4)	CDh–CFh	24

Note: The following register index numbers are reserved: C2h, C3h, and D0h through FFh.

Each I/O port 23h data transfer must be preceded by an I/O port 22h register 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 range, external I/O cycles also occur.

The CCR0 register (Table 2–11) determines if the 64K-byte memory area on 1M-byte boundaries and the 640K-byte to 1M-byte area are cacheable. This register also enables certain functions associated with cache control, suspend mode, and the clock-doubled mode.

Table 2–11. CCR0 Bit Definitions

Bit Position	Register Index	Description
0	NC0	Noncacheable 1M-byte boundaries: If 1, sets the first 64K bytes at each 1M-byte boundary as noncacheable.
1	NC1	Noncacheable upper memory area: If 1, sets 640K-byte to 1M-byte memory region noncacheable.
2	A20M	Enable A20M# pin: If 1, enables A20M#; otherwise pin is ignored.
3	KEN	Enable KEN# pin: If 1, enables KEN#; otherwise pin is ignored.
4	FLUSH	Enable FLUSH# pin: If 1, enables FLUSH#; otherwise pin is ignored.
5	BARB	Enable cache flush during hold: If 1, enables flushing of the internal cache when hold state is entered.
6	CKD	Enable clock double: If 1, enables clock-double mode. If 0, disables clock-double mode.
7	SUSP	Enable suspend pins: If 1, enables SUSP# and SUSPA#. If 0, SUSPA# floats; SUSP# is ignored.

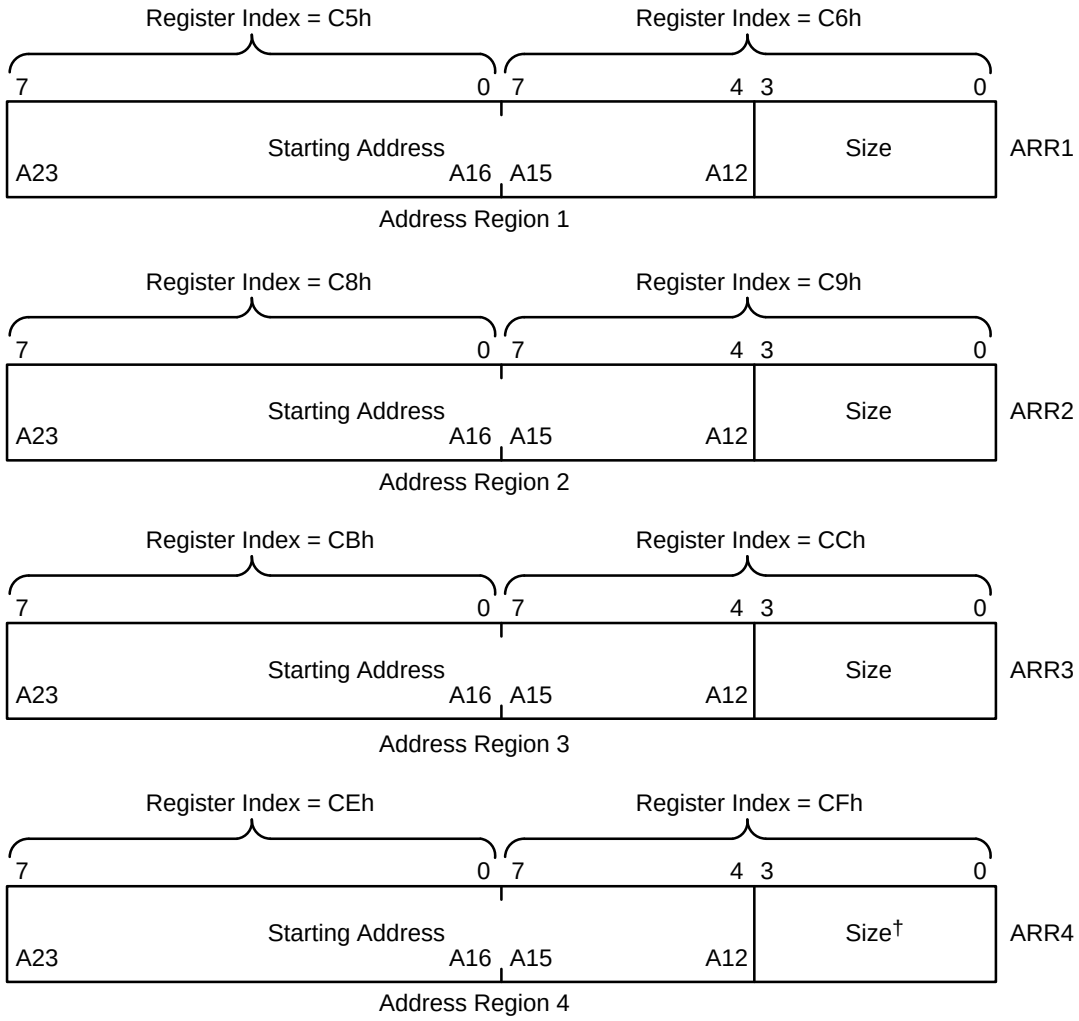
The CCR1 register (Table 2–12) sets up the internal cache operation and system-management mode (SMM). The ARR registers (Figure 2–15 on page 2-29, Figure 2–16 on page 2-30, and Table 2–9 and Table 2–10 on page 2-26) define the location and size of the memory regions associated with the internal cache. ARR1–ARR3 define three write-protected or noncacheable memory regions as designated by CCR1 bits WP1–WP3. ARR4 defines an SMM memory space as a noncacheable memory region when bit SM4 of CCR1 is set to 1. Other CCR1 bits enable SMM pins and control SMM memory access. The SMAC bit allows access to defined SMM space while not in an SMI service routine. The MMAC bit allows access to main memory that overlaps with SMM memory while in an SMI service routine for data access only.

Table 2–12. CCR1 Bit Definitions

Bit Position	Register Index	Description
0	—	Reserved
1	SMI	Enable SMM pins: If 1, SMI# and SMADS# are enabled. If 0, SMI# is ignored and SMADS# floats.
2	SMAC	System management memory access: If 1, noncode-segment prefixed data reads and writes to addresses within the SMM memory space cause external bus cycles to be issued with SMADS# active. SMI# is ignored. If 0, no effect on access.
3	MMAC	Main memory access: If 1, all noncode-segment prefixed data reads and writes that 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	WP1	Access region 1 control: If 1, region 1 is write protected and cacheable. If 0, region 1 is noncacheable.
5	WP2	Access region 2 control: If 1, region 2 is write protected and cacheable. If 0, region 2 is noncacheable.
6	WP3	Access region 3 control: If 1, region 3 is write protected and cacheable. If 0, region 3 is noncacheable.
7	SM4	Access region 4 control: If 1, region 4 is noncacheable SMM memory space. If 0, region 4 is noncacheable. SMI# input ignored.

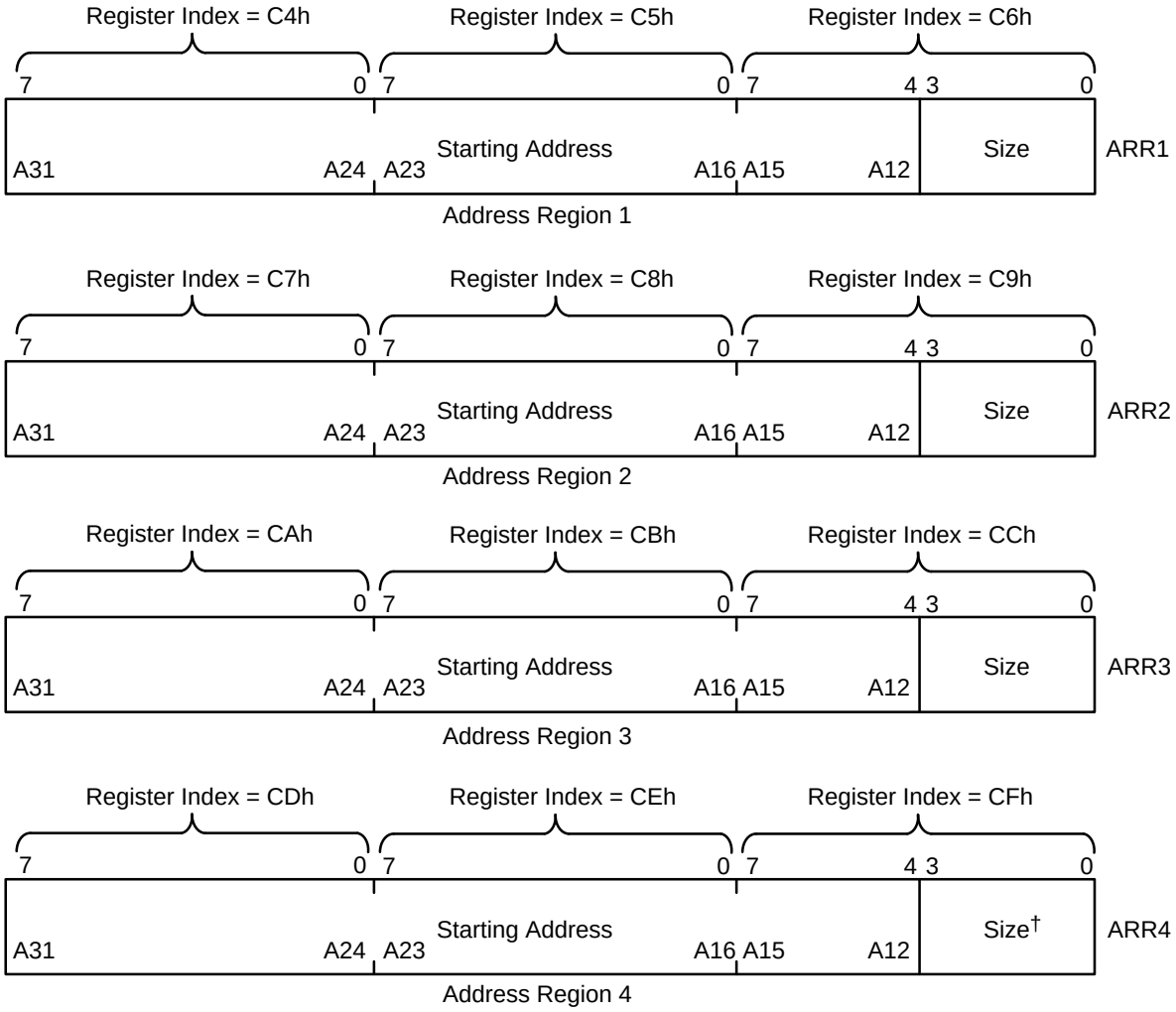
The ARR registers define address regions using a starting address and a block size. The noncacheable-region block sizes range from 4K bytes to 4G bytes (Table 2–13). A block size of zero disables the address region. The starting address of the address region must be on a block size boundary. For example, a 128K-byte block is allowed to have a starting address of 0K bytes, 128K bytes, 256K bytes, etc. The SMM memory region size is restricted to a maximum of 16M bytes. The block size must be defined for SMI# to be recognized.

Figure 2–15. TI486SXLC Address Region Registers (ARR1–ARR4)



†ARR4 (Size) must be 4K bytes to 16M bytes if ARR4 is defined as SMM memory space.

Figure 2–16. TI486SXL Address Region Registers (ARR1–ARR4)



†ARR4 (Size) must be 4K bytes to 16M bytes if ARR4 is defined as SMM memory space.

Table 2–13. ARR1–ARR4 Block Size Field

Bits 3–0	Block Size (Bytes)	Bits 3–0	Block Size (Bytes)
0h	Disabled	8h	512K
1h	4K	9h	1M
2h	8K	Ah	2M
3h	16K	Bh	4M
4h	32K	Ch	8M
5h	64K	Dh	16M
6h	128K	Eh	32M
7h	256K	Fh	4G

2.5.5 Debug Registers

Six Debug registers (DR0–DR3, DR6, and DR7), shown in Figure 2–17 and Figure 2–18, support debugging on the TI486SXL(C) family of microprocessors. 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 that can be executed only at privilege level 0.

Figure 2–17. TI486SXLC Debug Registers

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13	12 10	9	8	7	6	5	4	3	2	1	0	
LEN 3	R/W 3	LEN 2	R/W 2	LEN 1	R/W 1	LEN 0	R/W 0	0 0	GD	0 0 1	GE	LE	G3	L3	G2	L2	G1	L1	G0	L0	DR7
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0								BT	BS	0	0 1 1 1 1 1 1 1 1						B3	B2	B1	B0	DR6
Reserved																					DR5
Reserved																					DR4
Breakpoint 3 Linear Address																					DR3
Breakpoint 2 Linear Address																					DR2
Breakpoint 1 Linear Address																					DR1
Breakpoint 0 Linear Address																					DR0

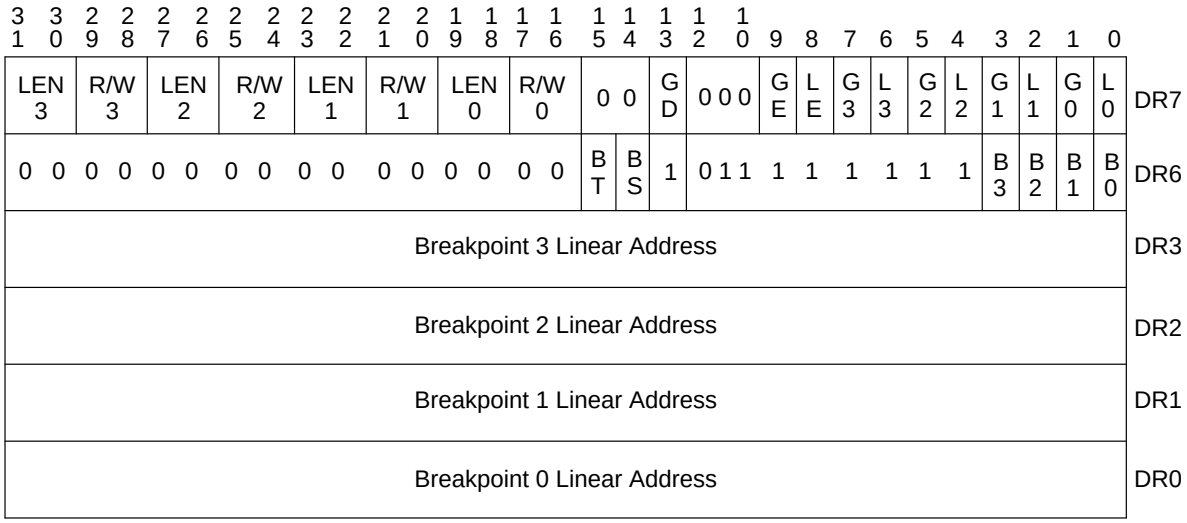
All bits marked as 0 or 1 are reserved and should not be modified.

The Debug Breakpoint Linear 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 execution as well as data-access breakpoints. Instruction-execution and data-access 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 2–14 lists the field definitions for the DR6 and DR7 registers.

Figure 2–18. TI486SXL Debug Registers



All bits marked as 0 or 1 are reserved and should not be modified.

Table 2–14. 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	Break applied to the DRi Breakpoint Linear 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	Length of the DRi Breakpoint Linear Address register: 00 – One-byte length 01 – Two-byte length 10 – Not used 11 – Four-byte length
	Gi	1	If set to 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 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	Globally disables Debug register access. GD bit is cleared whenever a debug exception occurs.

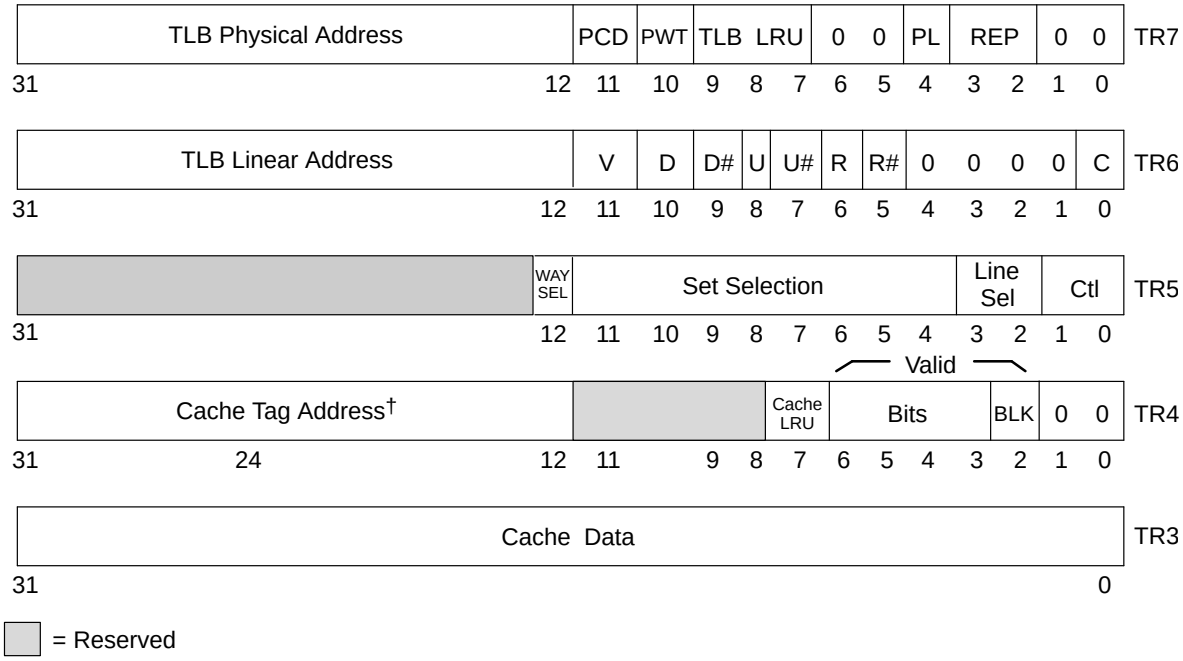
Code execution breakpoints can also be generated by placing the breakpoint instruction (INT3) at the location where control is to be regained. The single-step feature can be enabled by setting the TF flag in the EFLAGS register. This

causes the processor to perform a debug exception after the execution of every instruction.

2.5.6 Test Registers

The five Test registers, shown in Figure 2–19, test 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 2–15 and Table 2–16 list the bit definitions for the TR6 and TR7 registers.

Figure 2–19. Test Registers



† Bits 31–24 are reserved on the TI486SXLC.

2.5.6.1 TLB Test Registers

The microprocessor 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 are 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), two 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 lookup operations, the TLB data selected by the contents of TR6 is loaded into TR7.

Table 2–15. 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 only one match occurs in the TLB, the rest of the fields in TR6 and TR7 are updated according to the matching TLB entry. TLB write. A TLB entry is allocated to this linear address.
	11	Valid bit (V): TLB lookup. Always set to 1. 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 2–16.)
	8–7	User/supervisor attribute bit and its complement (U, U#). (Refer to Table 2–16.)
	6–5	Read/write attribute bit and its complement (R, R#). (Refer to Table 2–16.)
	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 is 1, set in which the tag was found. If PL is 0, undefined data. TLB write. If PL is 1, selects one of the four sets for replacement. If PL is 0, ignored.

Table 2–16. TR6 Attribute Bit Pairs

Bit (B)	Bit Complement (B#)	Effect on TLB Lookup	Effect on TLB Write
0	0	Do not match	Undefined
0	1	Match if the bit is 0	Clear the bit
1	0	Match if the bit is 1	Set the bit
1	1	Match if the bit is 1 or 0	Undefined

2.5.6.2 Cache Test Registers

The microprocessor on-chip cache is 8K bytes in size and is configured as two-way set-associative memory.

The cache memory is physically split into two 4K-byte blocks each containing 1024 lines. Associated with each 4K-byte block are 256 twenty-bit tags, which implies four lines in a block that are associated with the same tag. These four lines are consecutive at 16-byte boundaries. For each byte in a line, a valid bit indicates which of the four data bytes actually contains valid data. In addition, there is a valid bit associated with each block of four lines, which when reset, indicates that none of the 16-bytes in the four lines of that block contain valid data.

The LRU bit indicates which of the two sets was more recently accessed. The LRU bit is uninitialized for a given set after RESET or FLUSH#. The set's LRU bit remains uninitialized until the first read allocation to that set occurs. The first cache allocation to a given set is to way 1 and the LRU bit is then equal to 1. In a similar manner, the tag and valid bits of a given set and way are uninitialized until a read allocation occurs and the block valid bit is set.

The microprocessor contains three Test registers that allow testing of its internal cache. Using these registers, cache test writes and reads can be performed. Cache test writes cause the data in TR3 to be written to the selected way and entry in the cache. Cache test reads allow inspection of the data, the valid bits, and the LRU bit for the cache entry. For data to be written to the allocated entry, the valid bits for the entry must be set prior to the write of the data. Bit definitions for the cache Test registers are shown in Table 2–17.

Table 2–17. TR3–TR5 Bit Definitions

Register Name	Bit Position	Description
TR3	31–0	Cache data: Cache read. data accessed from the cache Cache write. to be written into the cache
TR4	31–12	Tag address: Cache read. tag address from which data is read Cache write. data written into the tag address of the selected set
	7	LRU: Cache read. the LRU bit associated with the cache set Cache write. ignored
	6–3	Valid bits: Cache read. four valid bits for the accessed line, (one bit per byte) Cache write. valid bits written into the line
	2	Block valid bit: Cache read. the block valid bit associated with the cache way Cache write. the block valid bit written into the selected way If 0, block is invalid (all 16 bytes are invalid). If 1, block is valid (one or more bytes may be valid in 16-byte line).
TR5	12	Way selection: If 0, way 0 is selected. If 1, way 1 is selected.
	11–4	Set selection. Selects one of 256 sets.
	3–2	Line selection. Selects one of four lines.
	1–0	Control bits. These bits control reading or writing the cache: If 00, ignored If 01, cache write If 10, cache read If 11, cache invalidate

2.6 Memory Address Space

The TI486SXLC directly addresses up to 16M bytes of physical memory, and the TI486SXL directly addresses up to 4G bytes 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.

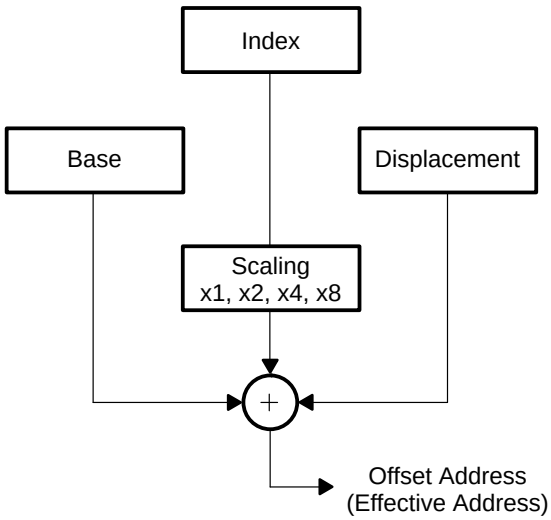
With the TI486SXL(C) microprocessor family, memory can be addressed using nine different addressing modes. These addressing modes are used to calculate an offset address often referred to as an effective address. Depending on the operating mode of the CPU, the offset is then combined using memory-management mechanisms to create and address a physical memory location.

Memory-management mechanisms on the microprocessor 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 microprocessor memory address space.

2.6.1 Offset Mechanism

The offset mechanism computes an offset (effective) address by summing up to three values: the base, the index, and the displacement. The base, if present, is the value in one of eight 32-bit General registers at the time the instruction is executed. The index, like the base, is a value that is determined from 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 of the memory address calculation is the displacement, which is a value of up to 32 bits in length supplied as part of the instruction. Figure 2–20 illustrates the calculation of the offset address.

Figure 2–20. Offset Address Calculation



Nine valid combinations of the base, index, scale factor, and displacement can be used with the TI486SXL(C) family instruction set. These combinations are listed in Table 2–18. The base and index both refer to contents of a register as indicated by [Base] and [Index].

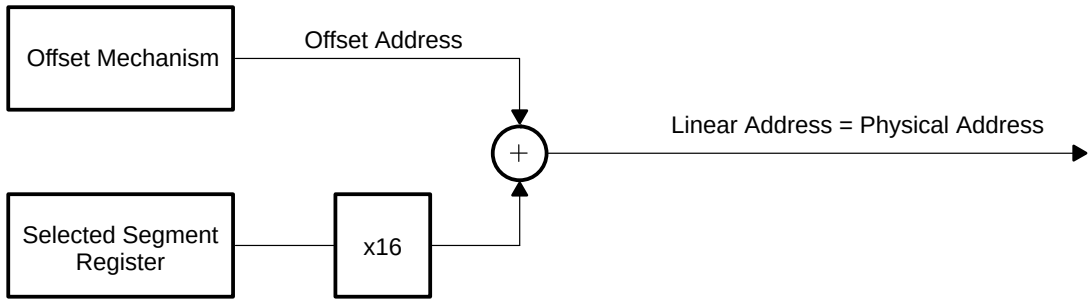
Table 2–18. Memory Addressing Modes

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

2.6.2 Real-Mode Memory Addressing

In real-mode operation, the TI486SXL(C) family of microprocessors address only the lowest 1M bytes (2^{20}) of memory. To calculate a physical memory address, the 16-bit segment base address located in the selected Segment register is shifted left by four bits and then the 16-bit offset address is added. For the TI486SXLC, the resulting 20-bit address is then extended with four zeros in the upper address bits to create the 24-bit physical address. For the TI486SXL, the resulting 20-bit address is then extended with 12 zeros in the upper address bits to create the 32-bit physical address. Figure 2–21 illustrates the real-mode address calculation. Address offsets larger than 65,535 cause a general protection fault. Physical addresses beyond 1M byte cause a segment-limit-overrun exception.

Figure 2–21. Real-Mode Address Calculation



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

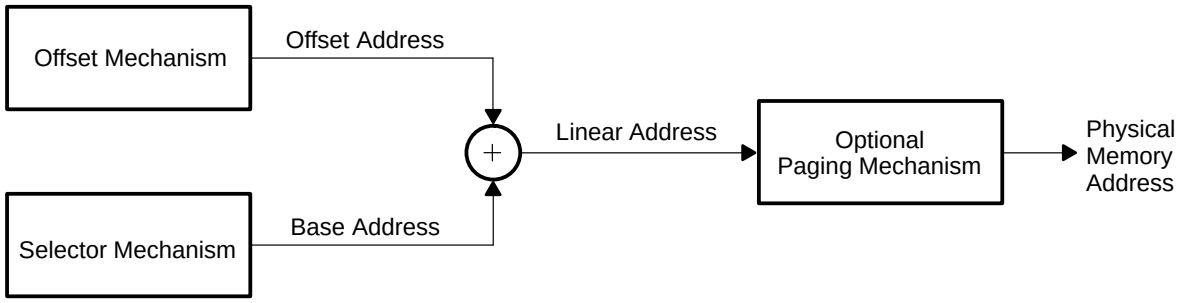
2.6.3 Protected-Mode Memory Addressing

In protected mode, three mechanisms calculate a physical memory address.

- 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 as illustrated in Figure 2–22. If paging is not used, the linear address is used as the physical memory address. If paging is enabled, the paging mechanism translates the linear address into the physical address. The offset mechanism, described earlier in this section, applies to both the real and protected modes. The selector and paging mechanisms are described in the following subsections.

Figure 2–22. Protected-Mode Address Calculation

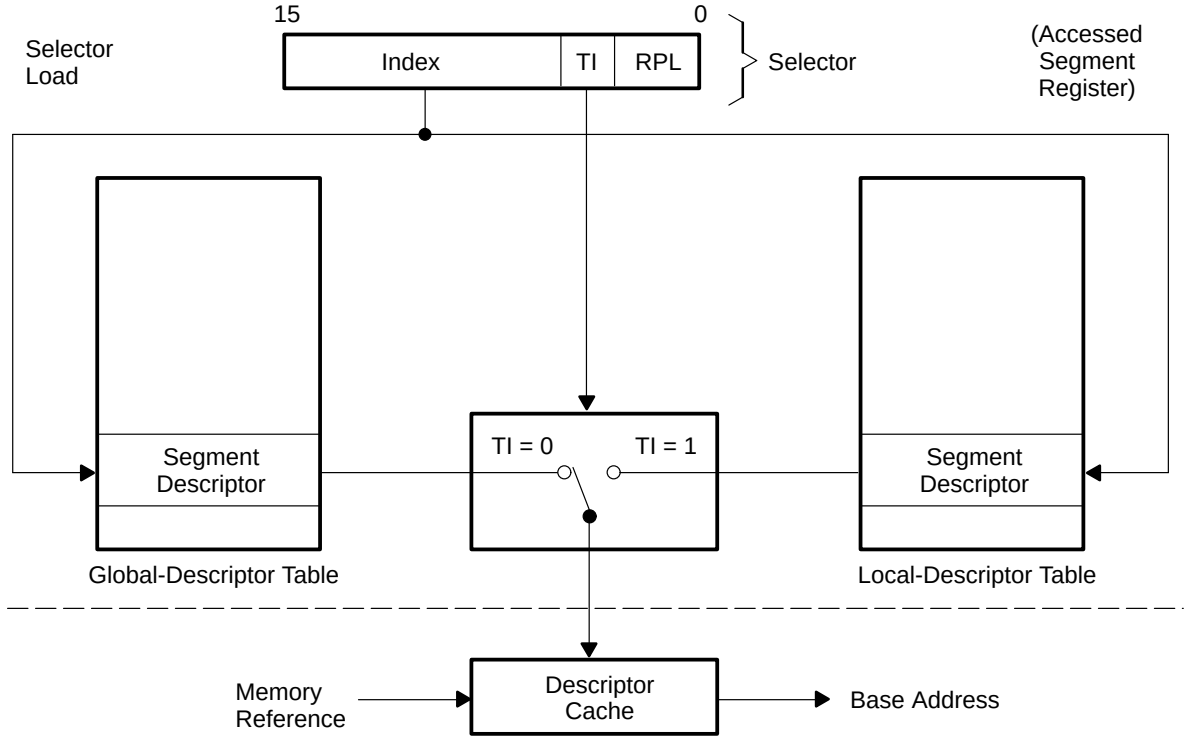


2.6.3.1 Selector Mechanism

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

The six Segment registers (CS, DS, SS, ES, FS, and GS) each contain a 16-bit selector. The selector 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, the limit, and the attributes of the selected segment and is cached on the microprocessor 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 2–23 illustrates the operation of the selector mechanism.

Figure 2–23. Selector Mechanism

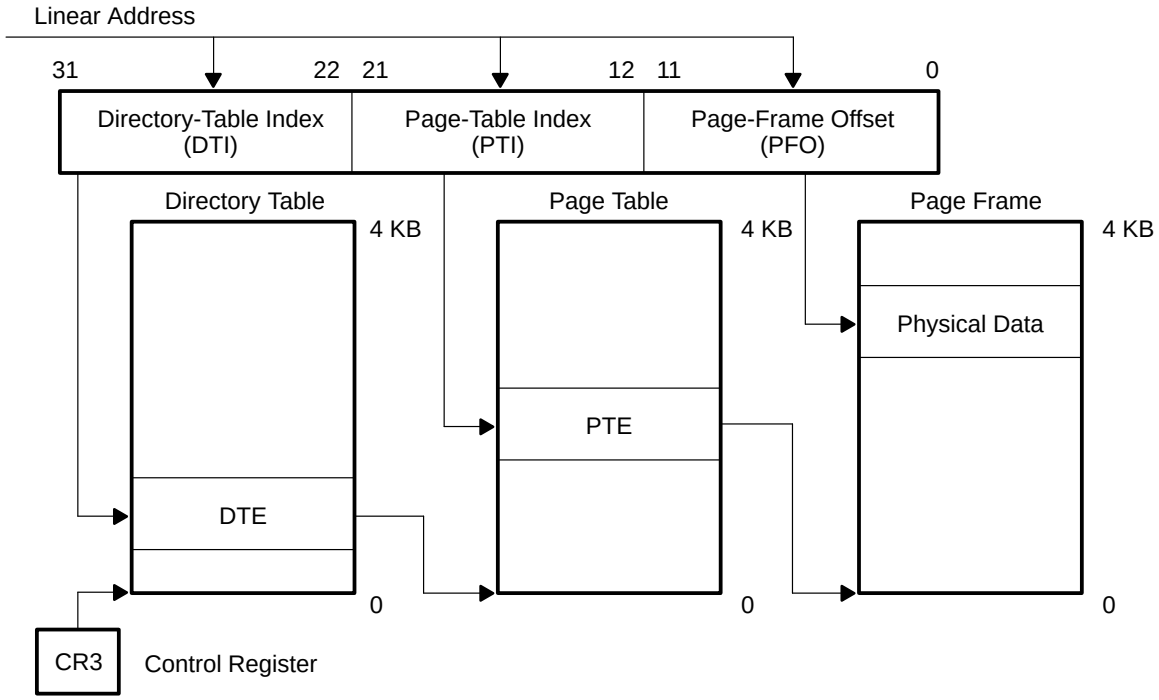


2.6.3.2 Paging Mechanism

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 always 4K bytes in size and are aligned to 4K-byte boundaries.

A page is addressed by using two levels of tables as illustrated in Figure 2–24. 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 master index of up to 1K individual 32-bit pointers to 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 4K-byte boundary. The physical address of the current page directory is stored in the CR3 Control register, also referred to as the Page-Directory Base register (PDBR).

Figure 2–24. Paging Mechanism



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 4K bytes in size and is itself a page. The lower 12 bits of the 32-bit linear address, referred to as the page-frame offset, locate the desired data within the page frame.

Since the page-directory table can point to 1K page tables, and each page table can point to 1K page frames, a total of 1M page frames can be implemented. Since each page contains 4K bytes, the microprocessor addresses up to 4G bytes of virtual memory 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 2–25 and listed in Table 2–19.

Figure 2–25. Directory- and Page-Table Entry (DTE and PTE) Format

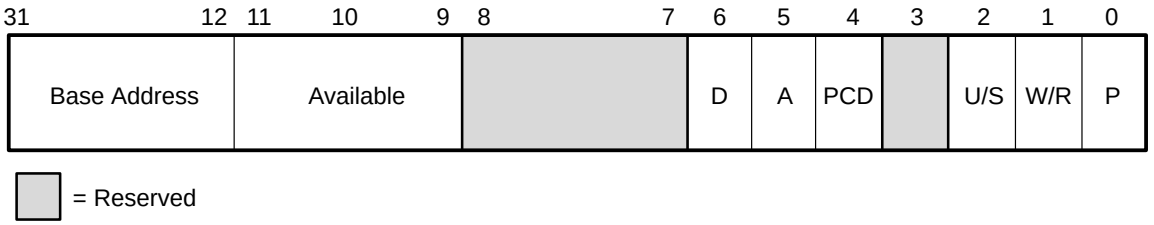


Table 2–19. Directory- and Page-Table Entry (DTE and PTE) Bit Definitions

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	—	Reserved and not available to the programmer
2	U/S	User/supervisor attribute. If set (user), page is accessible at all privilege levels. 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 that the programmer can use the remaining DTE/PTE bits.

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 and the operand is fetched. Both accessed bits (DTE and PTE) are set, 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 bit must be set to validate the remaining bits in the DTE and PTE. If either of the present bits is 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.

2.6.3.3 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 cache 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 128K bytes 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.

2.7 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.

All 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.

2.7.1 Interrupts

External events can interrupt normal program execution by using one of the three interrupt pins on the TI486SXL(C) family of microprocessors.

- Nonmaskable 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 usually 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 the execution of the IRET instruction, only one NMI rising edge is stored and then processed after execution of the next IRET.

During the NMI service routine, maskable interrupts are still 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 microprocessor 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 that is supplied by an external interrupt controller. This vector selects which of the 256 possible interrupt handlers is executed in response to the interrupt.

The SMM interrupt has higher priority than either the INTR or NMI. After SMI# is asserted, program execution is passed to an SMI service routine that runs in SMM address space reserved for this purpose. The remainder of this subsection (2.7.2, *Exceptions*, through 2.7.6, *Error Codes*, page 2-48) does not apply for SMM interrupts. SMM interrupts are described in Section 2.8, *System-Management Mode*, page 2-49.

2.7.2 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 that first caused the exception.

2.7.2.1 Trap Exceptions

A trap exception is reported immediately following the instruction that generated it. Trap exceptions are generated as follows:

- ❑ At a breakpoint
- ❑ By software interrupt instruction (INT0, INT3, INTn, BOUND)
- ❑ By a single-step operation
- ❑ By a data breakpoint

Software interrupts can be used to simulate hardware interrupts. For example, an INTn 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 INT3, or breakpoint-interrupt (vector 3), is a particular case of the INTn instruction. By inserting this one-byte instruction in a program, the user can set breakpoints in 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.

2.7.2.2 Fault Exceptions

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 that 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.

2.7.2.3 Abort Exceptions

An abort exception is a type of fault exception severe enough that the CPU cannot restart the program at the faulting instruction. Abort exceptions include the double fault (vector 8) and coprocessor segment overrun (vector 9).

2.7.3 Interrupt Vectors

When the CPU services an interrupt or exception, the current program's instruction pointer and flags are pushed onto the stack to let the interrupted program resume execution. 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.

2.7.3.1 Interrupt-Vector Assignments

Each interrupt (except SMI#) and each exception is assigned one of 256 interrupt-vector numbers (Table 2–20). The first 32 interrupt-vector assignments are defined or reserved. INT instructions acting as software interrupts can use any of the interrupt vectors 0 through 255. The nonmaskable hardware interrupt (NMI) is assigned vector 2.

In response to a maskable hardware interrupt (INTR), the microprocessor issues interrupt-acknowledge bus cycles to read the vector number from external hardware. These vectors should be in the vector range of 32–255 because vectors 0–31 are predefined.

2.7.3.2 Interrupt-Descriptor Table

The interrupt-vector number is used by the microprocessor 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 can contain different types of descriptors: interrupt gates, trap gates, and task gates. Interrupt gates are used mainly 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.

Table 2–20. Interrupt-Vector Assignments

Interrupt Vector	Function	Exception Type
0	Divide error	Fault
1	Debug exception	Trap/Fault (see Note)
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	Coprocessor segment overrun	Abort
10	Invalid TSS	Fault
11	Segment not present	Fault
12	Stack fault	Fault
13	General-protection fault	Fault
14	Page fault	Fault
15	Reserved	—
16	Coprocessor 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.

2.7.4 Interrupt and Exception Priorities

As the TI486SXL(C) family of microprocessors executes instructions, each follows a consistent policy for prioritizing exceptions and hardware interrupts as listed in Table 2–21. SMM interrupts always take precedence. Debug traps for the previous instruction and next instruction are handled in the next priority. When NMI and maskable INTR interrupts are both detected at the same instruction boundary, the microprocessor services the NMI interrupt first.

The microprocessor checks for exceptions in parallel with instruction decoding and execution. Several exceptions can result in 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 that way, exceptions can be serviced until the instruction executes properly.

The microprocessor supports restarting the instruction after all faults except when an instruction causes a task switch to a task whose task-state segment (TSS) is partially missing. A TSS can be partially missing if the TSS is not page-

aligned and one of the pages (where the TSS resides) is not currently in memory.

Table 2–21. Interrupt and Exception Priorities

Priority	Description	Notes
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	Nonmaskable 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, and 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	Coprocessor-error exception	Caused by ERROR# asserted
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	—

2.7.5 Exceptions in Real Mode

Many of the exceptions described in Table 2–20 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 2–22.

Table 2–22. 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, and/or GS segment limit overrun
14	Page fault	—

2.7.6 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 2–26 and the error-code bit definitions are listed in Table 2–23. Bits 15–3 (selector index) are not meaningful if the error code is generated as the result of a page fault. The error code is always zero for double faults and alignment-check exceptions.

Figure 2–26. Error-Code Format

Table 2–23. 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 = page not present 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 set, the exception occurred while trying to invoke exception or hardware interrupt handler.
Segment fault	Index of faulty selector	TI bit of faulty selector	0	If set, the exception occurred while trying to invoke exception or hardware interrupt handler.

2.8 System-Management Mode

System-management mode (SMM) provides an additional interrupt for system power management or software-transparent emulation of I/O peripherals. SMM is entered using the software-management interrupt (SMI#), which has a higher priority than any other interrupt including NMI. After receiving an SMI#, portions of the CPU state are automatically saved, SMM is entered, and program execution begins at the base of SMM space (Figure 2–27 and Figure 2–28). Running in protected SMM address space, the interrupt routine does not interfere with the operating system or any application program.

Seven SMM instructions have been added to the TI486SXL(C) microprocessor family instruction set that permit saving and restoring the total CPU state when in SMM mode. Two new pins, SMI# and SMADS#, support SMM functions.

Figure 2–27. TI486SXLC Memory and I/O Address Spaces

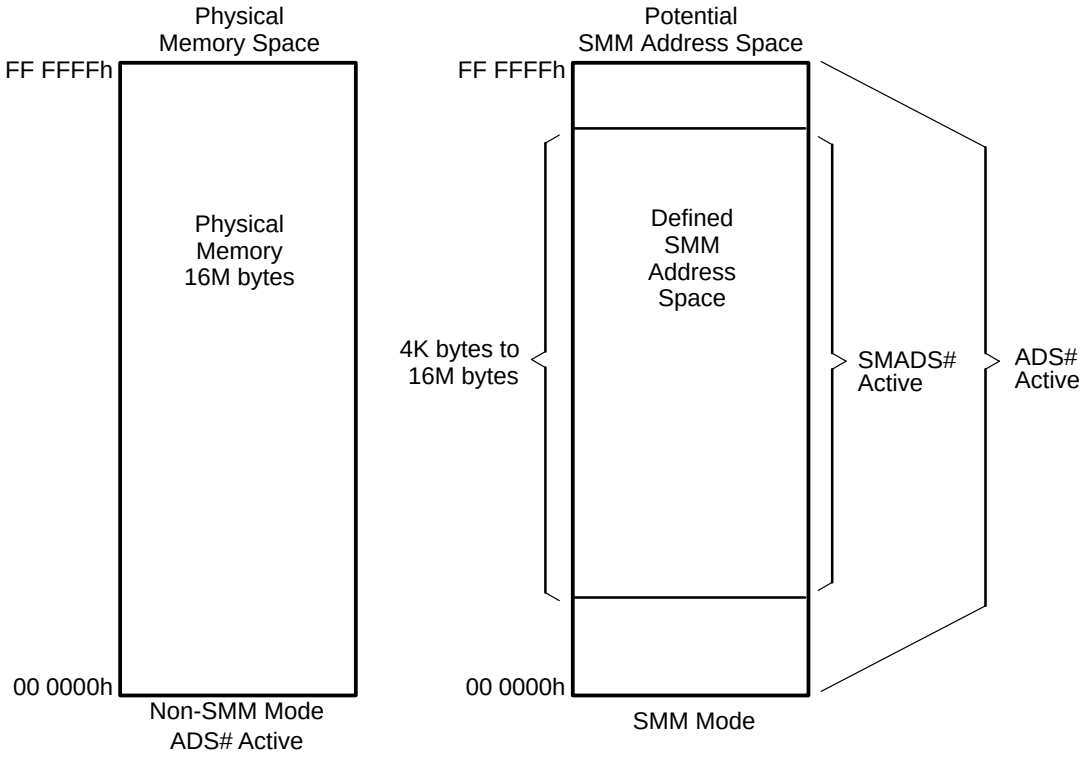
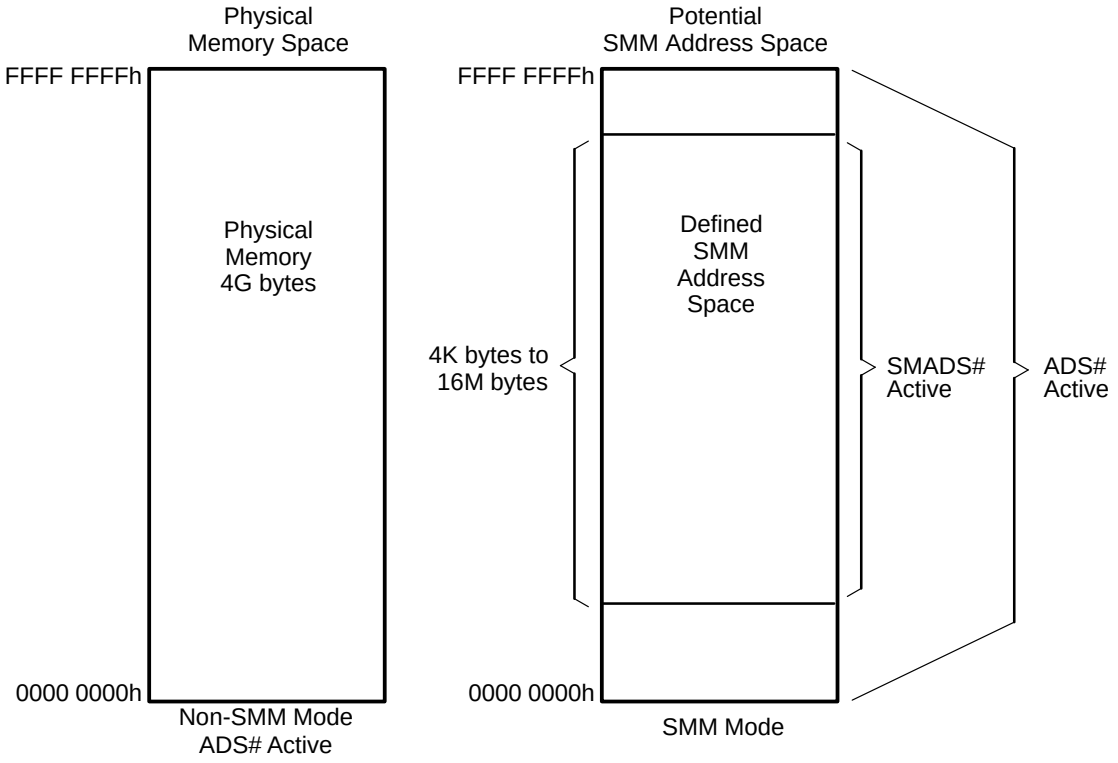


Figure 2–28. TI486SXL Memory and I/O Address Spaces



2.8.1 SMM Operations

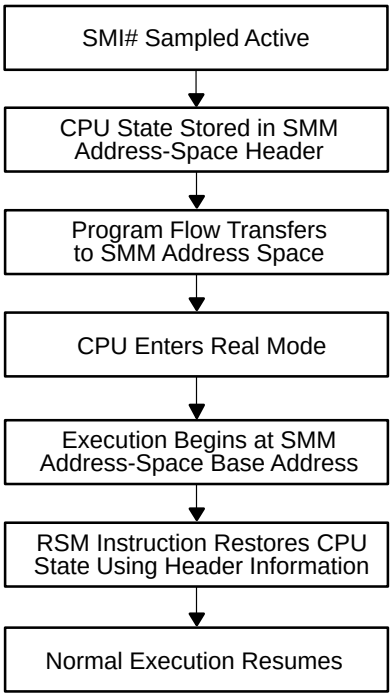
SMM operation is summarized in Figure 2–29. Entering SMM requires the assertion of SMI# for at least four CLK2 periods. For the SMI# input to be recognized, the following Configuration register bits must be set as follows:

SMI	CCR1(1)	= 1
SMAC	CCR1(2)	= 0
SM4	CCR1(7)	= 1
ARR4	SIZE(3–0)	> 0

The Configuration registers are discussed in Section 2.5, *System Register Set*, page 2-16. After recognizing SMI# 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 the 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 if the SMI service routine modifies registers other than those saved automatically, the complete CPU-state information must be saved.

Figure 2–29. SMM Execution Flow Diagram



A complete CPU-state save is performed by using MOV instructions to save normally accessible information and by using the SMM instructions to save CPU information that is normally inaccessible to the programmer. As will be explained, SMM instructions (SVDC, SVLDT, and SVTS) store the LDTR, TSR, and Segment registers and their associated descriptor cache entries in 80-bit memory locations. After power up or at the end of the SMI service routine, the MOV and additional SMM instructions (RSDC, RSLDT, and RSTS) restore the CPU state. The SMM RSM instruction returns the CPU to normal execution.

2.8.2 SMM Memory Space Header

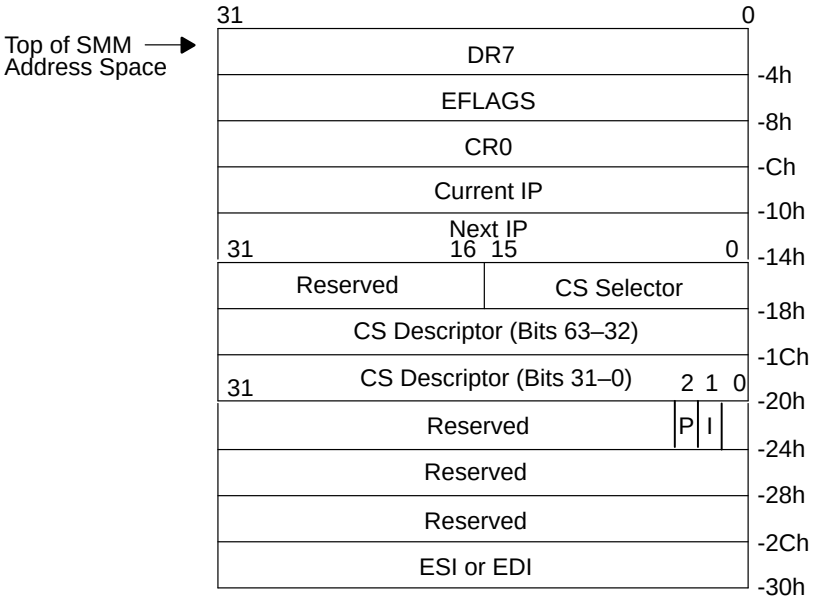
With every SMI interrupt, certain CPU-state information is automatically saved in the SMM memory space header located at the top of SMM address space (Table 2–24 and Figure 2–30). The header contains CPU-state information that is modified when servicing an SMI interrupt. Included in this information are two pointers. The current IP points to the instruction that is executing when the SMI is detected. The next IP points to the instruction that is executed after exiting SMM. The contents of the Debug register 7 (DR7), the extended Flag Word register (EFLAGS), and the Control register 0 (CR0) are also saved. 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 (Table 2–24) contain the same addresses, and the I and P fields contain valid information.

Table 2–24. SMM Memory Space Header

Name	Description	Size
DR7	The contents of Debug register 7	4 bytes
EFLAGS	The contents of the extended flag register	4 bytes
CR0	The contents of Control register 0	4 bytes
Current IP	The address of the instruction executed prior to servicing the SMI interrupt	4 bytes
Next IP	The address of the next instruction that is executed after exiting the SMM mode	4 bytes
CS Selector	Code Segment register selector for the current code segment	2 bytes
CS Descriptor	Code register descriptor for the current code segment	8 bytes
P	REP INSx/OUTSx† Indicator P is 1 if current instruction has a REP prefix P is 0 if current instruction does not have REP prefix	1 bit
I	IN, INSx, OUT, or OUTSx Indicator I is 1 if current instruction performed is an I/O WRITE I is 0 if current instruction performed is an I/O READ	1 bit
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

† INSx = INS, INSB, INSW, or INSD instruction, and OUTSx = OUTS, OUTSB, OUTSW, or OUTSD instruction.

Figure 2–30. SMM Memory Space Header



2.8.3 SMM Instructions

The TI486SXL(C) microprocessor family automatically saves the CPU-state information shown in Table 2–24 and Figure 2–30 when entering SMM. This 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 either modifies

more than what is saved in the SMM memory space header or forces the CPU to power down, the complete CPU-state information must be saved. Since the TI486SXL(C) microprocessors are static devices, their 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 2–25, can be executed only if: (a) the current privilege level (CPL) = 0 and the SMAC bit (CCR1, bit 2) are set; or (b) CPL = 0 and the CPU is in an SMI service routine (SMI# = 0). If both these conditions are not met and the microprocessor attempts to execute an SMM instruction, it generates an invalid-opcode exception. These instructions can be executed outside of defined SMM space provided the above conditions are met. All of the SMM instructions (except RSM) save or restore 80 bits of data, allowing the saved values to include the hidden portion of the register contents.

Table 2–25. 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 *(CS is automatically restored with RSM)*
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* Save 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
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.

† mem80 = 80-bit memory location.

2.8.4 SMM Memory Space

SMM memory space is defined by assigning address region 4 to SMM memory space. This assignment is made by setting bit 7 (SM4) in the on-chip CCR1 register. ARR4, also an on-chip Configuration register, specifies the base address and size of the SMM memory space. The base address must be a multiple of the SMM memory space size. For example, a 32K-byte SMM memory space must be located at a 32K-byte address boundary. The memory space size can range from 4K bytes to 16M bytes.

SMM memory space accesses can use address pipelining, and they are always noncacheable. SMM accesses ignore the state of the A20M# input 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 system-management access (SMAC) bit in the CCR1 register. This feature can 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 (noncode) accesses to main memory that overlap defined SMM memory space are allowed if bit 3 in CCR1 (MMAC) is set. In this case, ADS# strobes are generated for data accesses only, and SMADS# strobes continue to be generated for code accesses.

2.8.5 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 and the limits defined by the ARR4 register, and the SMI service routine begins execution at the SMM base address in real mode.

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

Hardware interrupts (INTRs and NMIs) can be serviced during an SMI service routine. If interrupts are to be serviced while operating in SMM memory space, the SMM memory space must be within the 0 to 1M-byte address range. This assures a proper return to the SMI service routine after handling the interrupt. INTRs are disabled automatically when entering SMM since the IF flag is set to its reset value. However, NMIs remain enabled. If you want to disable NMI, do it with the system hardware logic immediately after entering the SMI service routine.

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

To exit the SMI service routine, a resume (RSM) instruction, rather than an IRET, is executed. The RSM instruction causes the microprocessor to restore the CPU state using the SMM header information and resume execution at the interrupted point. If the programmer saved the full CPU state, reload the stored values using the MOV and the RSDC, RSLDT, and RSTS instructions before executing the RSM instruction.

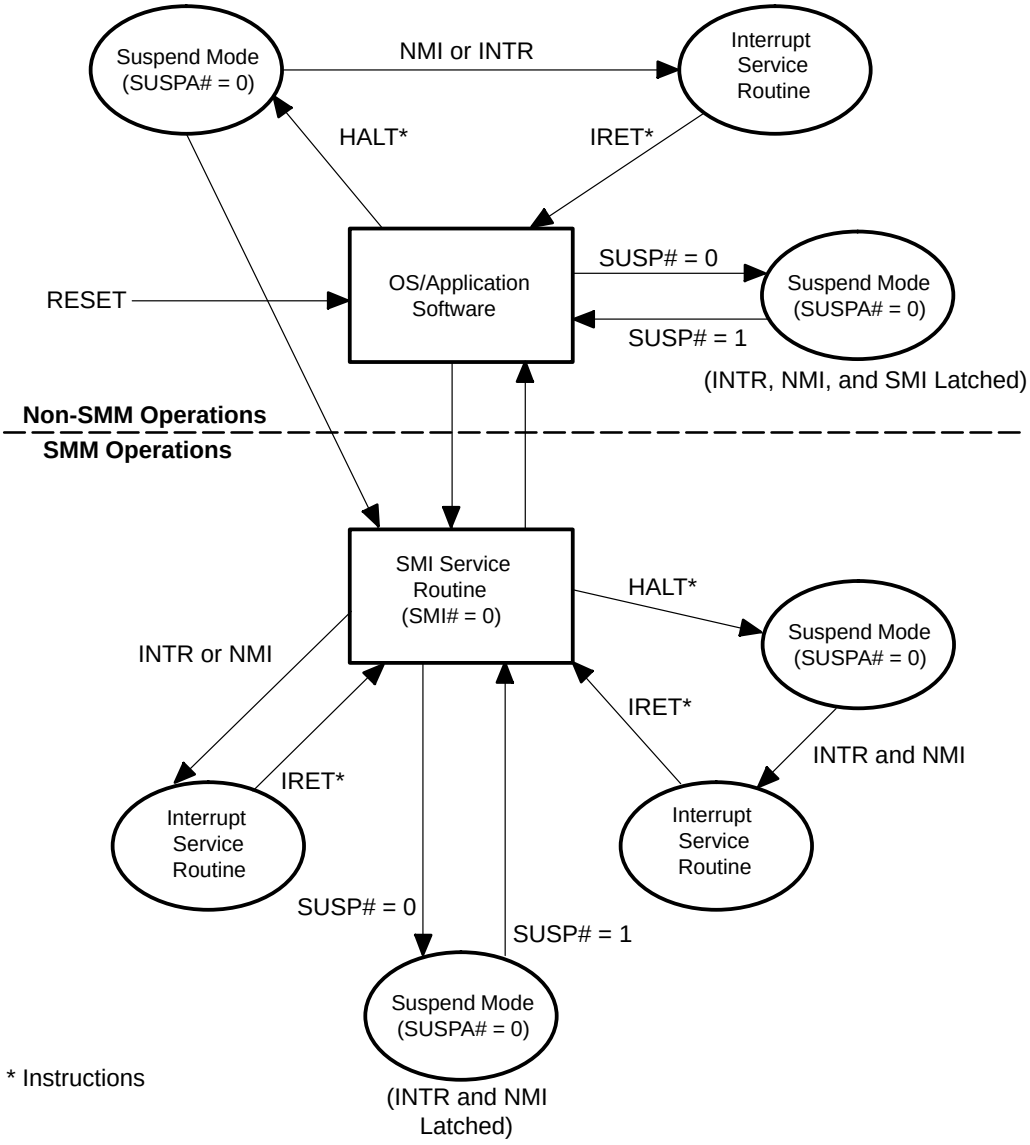
2.8.6 CPU States Related to SMM and Suspend Mode

The state diagram shown in Figure 2–31 illustrates the various CPU states associated with SMM and suspend mode. While in the SMI service routine, the TI486SXL(C) microprocessor family can enter suspend mode either by executing a HALT instruction or by asserting the SUSP# input.

During SMM operation and while in SUSP#-initiated suspend mode, an occurrence of either NMI or INTR is latched. 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 HALT 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#.

Figure 2–31. SMM and Suspended-Mode Flow Diagram



2.9 Shutdown and Halt

Shutdown occurs when a severe error is detected that prevents further processing. An NMI input causes the microprocessor to exit the shutdown mode 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 be exited only by a processor reset.

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

2.10 Protection

Segment protection and page protection are safeguards built into the TI486SXL(C) microprocessor family protected-mode architecture that 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 in subsection 2.6.3, *Protected-Mode Memory Addressing*, page 2-39. 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 privilege 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

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

2.10.1 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. It distinguishes between the privilege level of a routine actually accessing memory (CPL), and the privilege level of the original memory access requestor (RPL). The lower privilege level (0 is highest) of 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 ≥ 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 succeeds only if executed with an EPL ≥ 2.

2.10.2 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 be used unconditionally. 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 ≤ 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 bit map in the TSS to determine on a port-by-port basis whether I/O instructions (IN, OUT, INS, OUTS, REP INS, REP OUTS) are permitted. The remaining IOPL-sensitive operations generate a general-protection fault.

2.10.3 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.

2.10.3.1 Control Transfers

The five types of control transfers are summarized in Table 2–26. 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.

Table 2–26. 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	CALL	Call gate	GDT or LDT
Interrupt within task (could change CPL level)	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

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.

2.10.3.2 Gates

Gate descriptors provide protection for privilege transfers among executable segments. Gates change to routines of the same or higher privilege 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 higher than the gate descriptor's privilege level (DPL).

2.10.4 Initialization and Transition to Protected Mode

The TI486SXL(C) microprocessor family switches to real mode immediately after RESET and the system tables and registers are 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 that 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.

2.11 Virtual-8086 Mode

The TI486SXL(C) microprocessor family supports both real mode and virtual-8086 (V86) mode. V86 mode allows execution of 8086 applications and 8086 operating systems, yet still permits use of the TI486SXL(C) microprocessor-protection mechanism. V86 tasks run at privilege level 3. Upon entry, all segment limits are set to FFFFh (64K) as in real mode.

2.11.1 Memory Addressing

In V86 mode, Segment registers are used in the same manner as in real mode. The contents of the Segment register are shifted left four bits and added to the offset to form the segment base linear address. The TI486SXL(C) microprocessor family permits the operating system to select which programs use the V86 address mechanism and which programs use protected-mode addressing for each task.

The TI486SXL(C) microprocessor family also permits the use of paging when operating in V86 mode. Using paging, the 1M-byte address space of the V86 task can be mapped anywhere in the 4G-byte linear address space of the microprocessor CPU. As in real mode, linear addresses that exceed 1M byte cause a segment-limit-overrun exception.

The paging hardware allows multiple V86 tasks to run concurrently, provides protection, and isolates operating systems. 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 above 1M byte.

2.11.2 Protection

All V86 tasks operate at the lowest privilege level (level 3) and are subject to all of the microprocessor 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 is sensitive to the I/O privilege level (IOPL) than in protected mode. These instructions are CLI, INTn, IRET, POPF, PUSHF, and STI. The INT3, INT0 and BOUND variations of the INT instruction are not IOPL-sensitive.

2.11.3 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., a 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 can then choose to allow the 8086 operating system to handle the interrupt, or it can emulate the function of the interrupt han-

dler. 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.

2.11.4 Entering and Leaving V86 Mode

V86 mode is entered from protected mode either by 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 be exited only as the result of an interrupt or exception. The transition out must use a 32-bit trap or interrupt gate that must point to a nonconforming privilege level 0 segment (DPL = 0) or a 32-bit TSS. These restrictions are required to permit the trap handler to IRET to return to the V86 program.

Chapter 3

TI486SXLC Microprocessor Bus Interface

This chapter summarizes the TI486SXLC series processor signals and describes all inputs/outputs, functional timing and bus operations (including pipelined and nonpipelined addressing), various interfaces, and power management.

Topic **Page**

3.1 Input/Output Signals

This section describes the TI486SXLC series microprocessors' input and output signals. The discussion of these signals is arranged by the functional groups shown in Figure 3–1. Table 3–1 gives a brief description of each signal.

Figure 3–1. TI486SXLC Functional Signal Groupings

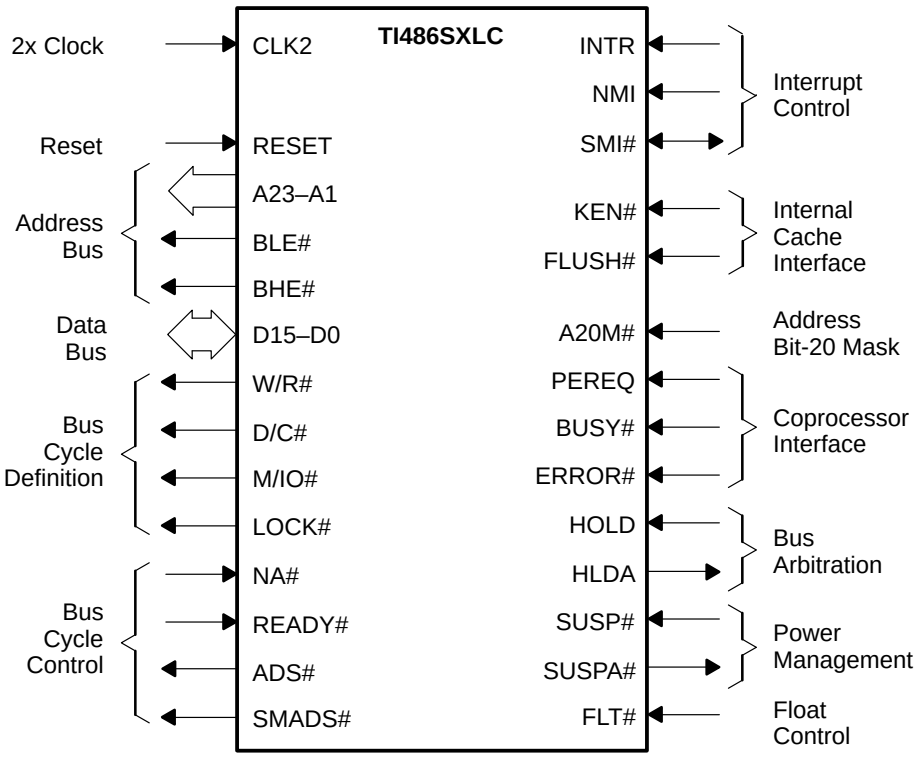


Table 3–1. TI486SXLC Signal Summary

Signal	Signal Name	Signal Group
ADS#	Address strobe	Bus-cycle control
A20M#	Address bit-20 mask	None
A23–A1	Address bus lines	Address bus
BHE#	Byte-high enable	Address bus
BLE#	Byte-low enable	Address bus
BUSY#	Processor extension busy	Coprocessor interface
CLK2	2X clock input	None
D15–D0	Data bus lines	None
D/C#	Data/control	Bus-cycle definition
ERROR#	Processor extension error	Coprocessor interface
FLT#	Float	None
FLUSH#	Cache flush	Internal cache interface
HLDA	Hold acknowledge	Bus arbitration
HOLD	Hold request	Bus arbitration
INTR	Maskable interrupt request	Interrupt control
KEN#	Cache enable	Internal cache interface
LOCK#	Bus lock	Bus-cycle definition
M/IO#	Memory/input-output	Bus-cycle definition
NA#	Next address request	Bus-cycle control
NMI	Nonmaskable interrupt request	Interrupt control
PEREQ	Processor extension request	Coprocessor interface
READY#	Bus ready	Bus-cycle control
RESET	Reset	None
SMADS#	SMM address strobe	Bus-cycle control
SMI#	System management interrupt	Interrupt control
SUSP#	Suspend request	Power management
SUSPA#	Suspend acknowledge	Power management
W/R#	Write/read	Bus-cycle definition

The following sections describe the signals and their functional characteristics. Additional signal information can be found in Chapter 5, *Electrical Specifications*. Chapter 5 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 TI486SXLC series microprocessors.

3.1.1 TI486SXLC Terminal Function Descriptions

Table 3–2 identifies and describes each of the TI486SXLC package terminals.

Table 3–2. TI486SXLC Terminal Functions

Terminal Name	Terminal No.	Description
A1	18	Address Bus (active high). The address bus (A23–A1) signals are 3-state outputs that provide addresses for physical memory and I/O ports. All address lines can be used to address physical memory, which allows a 16M-byte address space (00 0000h to FF FFFFh). During I/O port accesses, A23–A16 are driven low (except for coprocessor accesses). This permits a 64K-byte I/O address space (00 0000h to 00 FFFFh). During all coprocessor I/O accesses, address lines A22–A16 are driven low and A23 is driven high. This allows A23 to be used by external logic to generate a coprocessor select signal. Coprocessor command transfers occur with address 80 00F8h. Coprocessor data transfers occur with addresses 80 00FCh and 80 00FEh. A23–A1 float while the CPU is in a hold-acknowledge or float state.
A2	51	
A3	52	
A4	53	
A5	54	
A6	55	
A7	56	
A8	58	
A9	59	
A10	60	
A11	61	
A12	62	
A13	64	
A14	65	
A15	66	
A16	70	
A17	72	
A18	73	
A19	74	
A20	75	
A21	76	
A22	79	
A23	80	
ADS#	16	Address Strobe (active low). This 3-state output indicates that the TI486SXLC microprocessor has driven a valid address (A23–A1, BHE#, and BLE#) and bus-cycle definition (M/IO#, D/C#, and W/R#) on the appropriate output pins. During nonpipelined bus cycles, ADS# is active for the first clock of the bus cycle. During address pipelining, ADS# is asserted during the previous bus cycle and remains asserted until READY# is returned for that cycle. ADS# floats while the microprocessor is in a hold-acknowledge or float state.
A20M#	31	Address Bit-20 Mask (active low). This input causes the microprocessor to mask (force low) physical address bit 20 when driving the external address bus or performing an internal cache access. When the processor is in real mode, asserting A20M# emulates the 1M-byte address wraparound that occurs on the 8086. The A20 signal is never masked when paging is enabled regardless of the state of the A20M# input. The A20M# input is ignored following reset and can be enabled using the A20M bit in the CCR0 Configuration register. A20M# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.

Table 3–2. TI486SXLC Terminal Functions (Continued)

Terminal Name	Terminal No.	Description
BHE# BLE#	19 17	Byte Enables (active low). Byte-low enable (BLE#) and byte-high enable (BHE#) 3-state outputs indicate which byte(s) of the 16-bit data bus are selected for data transfer during the current bus cycle. BLE# selects the low byte (D7–D0) and BHE# selects the high byte (D15–D8). When BHE# and BLE# are asserted, both bytes (all 16 bits) of the data bus are selected. BLE# and BHE# float while the CPU is in a hold-acknowledge or float state. BHE# = BLE# = 1 never occurs during a bus cycle.
BUSY#	34	Coprocessor Busy (active low). This input indicates to the TI486SXLC that the coprocessor is currently executing an instruction and is unable to accept another opcode. When the microprocessor encounters a WAIT instruction or any coprocessor instruction that operates on the coprocessor stack (i.e., load, pop, or arithmetic operation), BUSY# is sampled. BUSY# is continually sampled and must be recognized as inactive before the CPU supplies the coprocessor with another instruction. However, coprocessor instructions FNINIT and FNCLEX are allowed to execute even if BUSY# is active because they are used for coprocessor initialization and exception clearing. BUSY# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.
CLK2	15	2X Clock Input (active high). This input signal is the basic timing reference for TI486SXLC microprocessors. The CLK2 input is internally divided by two to generate the internal processor clock. The external CLK2 is synchronized to a known phase of the internal processor clock by the falling edge of the RESET signal. External timing parameters are defined with respect to the rising edge of CLK2. For the TI486SXLC2 microprocessors, the CLK2 input is used internally to generate the internal core processor clock and the internal bus interface clock. The external CLK2 is synchronized to a known phase of the internal processor clock by the falling edge of the RESET signal. External timing parameters are defined with respect to the rising edge of CLK2.
D/C#	24	Data/Control. This 3-state, bus-cycle-definition signal is low during control cycles and is high during data cycles. Control cycles are issued during functions such as a halt instruction, interrupt servicing, and code fetching. Data bus cycles include data access from either memory or I/O.

Table 3–2. TI486SXLC Terminal Functions (Continued)

Terminal Name	Terminal No.	Description
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15	1 100 99 96 95 94 93 92 90 89 88 87 86 83 82 81	Data Bus (active high). The data bus signals (D15–D0) are 3-state bidirectional signals that provide the data path between the microprocessor, the external memory, and the I/O devices. The data-bus inputs receive data during memory-read, I/O-read, and interrupt-acknowledge cycles and delivers output 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 float while the CPU is in a hold-acknowledge or float state.
ERROR#	36	Coprocessor Error (active low). This input indicates that the coprocessor generated an error during execution of an instruction. ERROR# is sampled by the microprocessor whenever a coprocessor instruction is executed. If ERROR# is sampled active, the processor generates exception 16, which is then serviced by the exception handling software. The following coprocessor instructions, which clear coprocessor error flags and save the coprocessor state, do not generate an exception 16 even if ERROR# is active: FNINIT, FNCLEX, FNSTSW, FNSTCW, FNSTENV, FNSAVE. ERROR# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.
FLT#	28	Float (active low). This input forces all bidirectional and output signals to a 3-state condition. Floating the signals allows the microprocessor signals to be driven externally without physically removing the device from the circuit. The microprocessor must be reset following assertion or negation of FLT#. Use FLT# only for testing. FLT# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.
FLUSH#	30	Cache Flush (active low). This input invalidates (flushes) the entire cache. Use of FLUSH# to maintain cache coherency is optional. The cache may also be invalidated during each hold-acknowledge cycle by setting the BARB bit in the CCR0 Configuration register. The FLUSH# input is ignored following reset and can be enabled using the FLUSH bit in the CCR0 Configuration register. FLUSH# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.

Table 3–2. TI486SXLC Terminal Functions (Continued)

Terminal Name	Terminal No.	Description
HOLD	4	Hold Request (active high). This input indicates that another bus master requests control of the local bus. The bus arbitration (HOLD and HLDA) signals allow the microprocessor to relinquish control of its local bus when requested by another bus master device. Once the processor has relinquished its 3-stated bus, the bus master device can then drive the local bus signals. After recognizing the HOLD request and completing the current bus cycle or sequence of locked bus cycles, the microprocessor responds by floating the local bus and asserting the hold-acknowledge (HLDA) output. Once HLDA is asserted, the bus remains granted to the requesting bus master until HOLD becomes inactive. When the microprocessor recognizes that HOLD is inactive, it simultaneously drives the local bus and drives HLDA inactive. External pullup resistors may be required on some of the microprocessor 3-state outputs to ensure that they remain inactive while in a hold-acknowledge state. The HOLD input is not recognized while RESET is active. If HOLD is asserted while RESET is active, RESET has priority, and the microprocessor places the bus into an idle state instead of a hold-acknowledge state. The HOLD input is also recognized during suspend mode provided that the CLK2 input has not been stopped. HOLD is level-sensitive and must meet specified setup and hold times for correct operation.
HLDA	3	Hold Acknowledge (active high). This output indicates that the microprocessor is in a hold-acknowledge state and has relinquished control of its local bus. While in the hold-acknowledge state, the microprocessor drives HLDA active and continues to drive SUSPA#, if enabled. The other microprocessor outputs are in the high-impedance state, allowing the requesting bus master to drive these signals. If the on-chip cache can satisfy bus requests, the microprocessor continues to operate during hold-acknowledge states. A20M# is internally recognized during this time. The microprocessor deactivates HLDA when the HOLD request is driven inactive. The microprocessor stores an NMI rising edge during a hold-acknowledge state for processing after HOLD is inactive. The FLUSH# input is also recognized during a hold-acknowledge state. If SUSP# is asserted during a hold-acknowledge state, the microprocessor may or may not enter suspend mode depending on the state of the internal execution pipeline. Table 3–3 summarizes the state of the microprocessor signals during hold acknowledge.
INTR	40	Maskable Interrupt Request. This level-sensitive input 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 Flag Word register IF bit. When unmasked, the microprocessor responds to the INTR input by issuing two locked interrupt-acknowledge cycles. To assure recognition of the INTR request, INTR must remain active until the start of the first interrupt-acknowledge cycle.

Table 3–2. TI486SXLC Terminal Functions (Continued)

Terminal Name	Terminal No.	Description
KEN#	29	Cache Enable (active low). This input indicates that the data returned during the current cycle is cacheable. When KEN# is active and the microprocessor performs a cacheable code-fetch or memory-data-read cycle, the cycle is transformed into a cache fill. Use of the KEN# input to control cacheability is optional. The Noncacheable Region registers can also control cacheability. Memory addresses specified by the Noncacheable Region registers cannot be cached regardless of the state of KEN#. I/O accesses, locked reads, SMM address space accesses, and interrupt-acknowledge cycles are never cached. During cached code fetches, two contiguous read cycles are performed to completely fill the 4-byte cache line. KEN# must be asserted during both read cycles to cause a cache line fill. During memory data reads, the microprocessor performs as many read cycles as necessary to supply the required data to complete the current operation. Valid bits are maintained for each byte in the cache line and each block of four lines, thus allowing data operands of less than four bytes to reside in the cache. If two read cycles are performed with the same address (A23–A2), KEN# must be asserted during both cycles to cache the data in these cycles. If the data is cached, the microprocessor ignores the state of the byte enables (BHE# and BLE#), and all data on the bus is cached. The KEN# input is ignored following reset and can be enabled using the KEN bit in the CCR0 Configuration register. KEN# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.
LOCK#	26	LOCK (active low). This 3-state, bus-cycle-definition signal is asserted to deny access to the CPU bus by other bus masters. The LOCK# signal may be explicitly activated during bus operations by including the LOCK prefix on certain instructions. LOCK# is always asserted during descriptor and page table updates, interrupt-acknowledge sequences, and when executing the XCHG instruction. The microprocessor does not enter the hold-acknowledge state in response to HOLD while the LOCK# output is active.
M/IO#	23	Memory/IO. This 3-state, bus-cycle-definition signal is low during I/O read and write cycles and is high during memory cycles.
NA#	6	Next Address Request (active low). This input requests address pipelining by the system hardware. When asserted, the system indicates that it is prepared to accept new bus-cycle definition and address signals (M/IO#, D/C#, W/R#, A23–A1, BHE#, and BLE#) from the microprocessor even if the current bus cycle has not been terminated by assertion of READY#. If the microprocessor has an internal bus request pending and the NA# input is sampled active, the next bus-cycle definition and address signals are driven onto the bus.
NC†	27, 45, 46	Make no external connection.

† Connecting or terminating (high or low) any NC terminal(s) may cause the microprocessor to produce unpredictable results or not operate.

Table 3–2. TI486SXLC Terminal Functions (Continued)

Terminal Name	Terminal No.	Description
NMI	38	Nonmaskable Interrupt Request. This rising-edge-sensitive input causes the processor to suspend execution of the current instruction stream and begin execution of an NMI interrupt service routine. The NMI interrupt service request cannot be masked by software. Asserting NMI causes an interrupt that internally supplies interrupt vector 2h to the CPU core. External interrupt-acknowledge cycles are not necessary since the NMI interrupt vector is supplied internally. Once NMI processing has started, no additional NMIs are processed until an IRET instruction is executed. The microprocessor samples NMI at the beginning of each phase two (ϕ2) clock period. To assure recognition, NMI must be inactive for at least eight CLK2 periods and then be active for at least eight CLK2 periods. Additionally, specified setup and hold times must be met to assure recognition at a particular clock edge.
PEREQ	37	Coprocessor Request (active high). This input indicates that the coprocessor is ready to transfer data to or from the CPU. The coprocessor can assert PEREQ in the process of executing a coprocessor instruction. The microprocessor internally stores the current coprocessor opcode and transfers the correct data to support coprocessor operations. The microprocessor employs PEREQ to synchronize the transfer of required operands. PEREQ is internally connected to a pulldown resistor to prevent this signal from floating active when left unconnected.
READY#	7	Ready (active low). This input is generated by the system hardware to indicate that the current bus cycle can be terminated. During a read cycle, assertion of READY# indicates that the system hardware has presented valid data to the CPU. When READY# is sampled active, the microprocessor latches the input data and terminates the cycle. During a write cycle, READY# assertion indicates that the system hardware has accepted the microprocessor output data. READY# must be asserted to terminate every bus cycle, including halt and shutdown indication cycles.
RESET	33	Reset (active high). When asserted, RESET suspends all operations in progress and places the microprocessor into a reset state. RESET is a level-sensitive synchronous input and must meet specified setup and hold times to be properly recognized by the microprocessor. The microprocessor begins executing instructions at physical address location FF FFF0h approximately 400 CLK2 edges after RESET is driven inactive (low). While RESET is active, the microprocessor is initialized to nonclock-doubled mode (for the TI486SXLC2). All other input pins except FLT# are ignored. The remaining signals are initialized to their reset state during the internal processor reset sequence. The reset signal states for the microprocessor are shown in Table 3–3.
SMADS#	20	SMM Address Strobe (active low). SMADS#, a 3-state output, is asserted instead of the ADS# during SMM bus cycles.This 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.

Table 3–2. TI486SXLC Terminal Functions (Continued)

Terminal Name	Terminal No.	Description
SMI#	47	System Management Interrupt (active low). This 3-state, bidirectional, level-sensitive input/output signal is an interrupt with higher priority than the NMI interrupt. SMI# must be active for at least four CLK2 clock periods to be recognized by the microprocessor. After the SMI is acknowledged, the SMI# pin is driven low by the microprocessor 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. SMI# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.
SUSP#	43	Suspend Request (active low). This input requests the microprocessor to enter suspend mode. After recognizing SUSP# as active, the processor completes execution of the current instruction, any pending decoded instructions, and associated bus cycles. In addition, the microprocessor waits for the coprocessor to indicate a not-busy status (BUSY# = 1) before entering suspend mode and asserting suspend acknowledgement (SUSPA#). SUSP# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.
SUSPA#	44	Suspend Acknowledge (active low). This output indicates that the microprocessor has entered the suspend mode as a result of SUSP# assertion or execution of a HALT instruction.
V_{CC}	8 9 10 21 32 39 42 48 57 69 71 84 91 97	5-V Power Supply. All pins must be connected and used.

Table 3–2. TI486SXLC Terminal Functions (Continued)

Terminal Name	Terminal No.	Description
V_{SS}	2 5 11 12 13 14 22 35 41 49 50 63 67 68 77 78 85 98	Ground Pins. All pins must be connected and used.
W/R#	25	Write/Read. This 3-state, bus-cycle-definition signal is low during read cycles (data is read from memory or I/O) and is high during write bus cycles (data is written to memory or I/O).

3.1.2 Signal States During Reset and Hold Acknowledge

RESET is the highest priority input signal. When RESET is asserted, the microprocessor aborts any current bus cycle and establishes real-mode bus-cycle definition with active buses. See Table 3–3 and Section 3.3, *Reset Timing and Internal Clock Synchronization*, page 3-17.

The microprocessor enters the hold-acknowledge state in response to assertion of the HOLD input. During the hold-acknowledge state, the microprocessor floats all output and bidirectional signals except for HLDA and SUSPA#. In the hold-acknowledge state, all inputs except HOLD, FLUSH#, FLT#, SUSP# and RESET are ignored. See Table 3–3 and subsection 3.4.8, *Hold Acknowledge State*, page 3-39. The hold-acknowledge state lets an external device acquire the system bus.

Table 3–3. TI486SXLC Signal States During Reset and Hold Acknowledge

Signal Name	Signal State During Reset	Signal State During Hold Acknowledge
A20M#	Ignored	Input recognized
A23–A1	1	Float
ADS#	1	Float
BHE#, BLE#	0	Float
BUSY#	Initiates self test	Ignored
D15–D0	Float	Float
D/C#	1	Float
ERROR#	Ignored	Ignored
FLT#	Input recognized	Input recognized
FLUSH#	Ignored	Input recognized
HLDA	0	1
HOLD	Ignored	Input recognized
INTR	Ignored	Input recognized
KEN#	Ignored	Ignored
LOCK#	1	Float
M/IO#	0	Float
NA#	Ignored	Ignored
NMI	Ignored	Input recognized
PEREQ	Ignored	Ignored
READY#	Ignored	Ignored
RESET	Input recognized	Input recognized
SMADS#	Float	Float
SMI#	Ignored	Input recognized
SUSP#	Ignored	Input recognized
SUSPA#	Float	Driven
W/R#	0	Float

3.2 Bus-Cycle Definition

The bus-cycle-definition signals consist of four 3-state outputs (M/IO#, D/C#, W/R#, and LOCK#) that define the type of bus-cycle operation. Table 3–4 defines the bus cycles for the possible states of these signals. M/IO#, D/C#, and W/R# are the primary bus-cycle-definition signals and are driven valid as ADS# (address strobe) becomes active. During nonpipelined cycles, the LOCK# output is driven valid along with M/IO#, D/C#, and W/R#. During pipelined addressing, LOCK# is driven at the beginning of the bus cycle, which is after ADS# becomes active for that cycle. The bus-cycle-definition signals are active low and float while the microprocessor is in a hold-acknowledge or float state.

Table 3–4. TI486SXLC 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	X	—
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	Halt: A23–A1=2h, BHE#=1, and BLE#=0 Shutdown: A23–A1=0h, BHE#=1, and BLE#=0
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

3.2.1 Clock Doubling Using Software Control

The clock-doubled feature of the TI486SXLC2 is enabled/disabled using Configuration Control register 0 (CCR0), bit 6. The following software sets and resets CKD:

Set CKD programming sequence:

```
mov   al, 0C0h          ;select CCR0
out   22h, al
in    al, 23h           ;read CCR0
mov   ah, al            ;save in AH
or    ah, 40h           ;set AH<6>
mov   al, 0C0h          ;select CCR0
out   22h, al
mov   al, ah
out   23h, al           ;write CCR0
```

Reset CKD programming sequence:

```
mov   al, 0C0h          ;select CCR0
out   22h, al
in    al, 23h           ;read CCR0
mov   ah, al            ;save in AH
and   ah, 0BFh          ;reset AH<6>
mov   al, 0C0h          ;select CCR0
out   22h, al
mov   al, ah
out   23h, al           ;write CCR0
```

3.2.1.1 Entering Clock-Doubled Mode

The TI486SXLC2 microprocessors power up in the nonclock-doubled mode. To enter the clock-doubled mode, set CLK2 to the desired frequency inside the phase-locked loop (PLL) lock range (see Table 5–5 and Table 5–6) and issue the set-CKD-programming sequence. Approximately 20 μs after the final OUT instruction has exited the processor pipeline, the PLL locks and the CPU enters clock-doubled mode. Until the PLL is locked, the processor continues to operate in the nonclock-doubled mode.

3.2.1.2 Clock-Scaling Sequence

To scale or stop CLK2 input when the processor is in clock-doubled mode, issue the reset-CKD-programming sequence. The final OUT instruction exiting the processor pipeline resets the CKD bit and puts the microprocessor into nonclock-doubled mode. This must occur before scaling or stopping the CLK2 input to prevent a synchronization error. This may be ensured by issuing a JUMP instruction, such as JMP $+2, before scaling CLK2.

To return the processor to clock-doubled mode, set CLK2 to the desired frequency inside the PLL lock range and issue the set-CKD-programming sequence. Approximately 20 μs after the final OUT instruction has exited the processor pipeline, the PLL locks and the processor enters clock-doubled mode.

3.2.1.3 Suspend Mode

Suspend mode can be initiated when the TI486SXLC2 microprocessor is in clock-doubled mode as long as the CLK2 input is not scaled or stopped. Suspend mode does not disable the PLL; instead, changing the CLK2 frequency causes the PLL to lose lock.

For more detailed information on entering and exiting suspend in nonclock-doubled mode, refer to subsection 3.2.2, *Power Management.*

To get the lowest possible power state, bring the microprocessor out of clock-doubled mode, enter the suspend mode (using software or hardware), and stop the CLK2 input.

3.2.2 Power Management

The power-management signals allow the TI486SXLC series microprocessors to enter suspend mode. Suspend-mode circuitry allows the microprocessor to consume minimal power while maintaining the entire internal CPU state.

3.2.2.1 Suspend Request (SUSP#)

Suspend request (SUSP#) is an active-low input that requests the TI486SXLC series microprocessors to enter suspend mode. For TI486SXLC2 microprocessors, follow the clock-scaling sequence procedure in subsection 3.2.1.2 to enter nonclock-doubled mode before scaling or stopping the CLK2 input.

After recognizing SUSP# is active, the processor completes execution of the current instruction, any pending decoded instructions, and associated bus cycles. In addition, the microprocessor waits for the coprocessor to indicate a not-busy condition (BUSY#=1) before entering suspend mode and asserting suspend acknowledge (SUSPA#). During suspend mode, internal clocks are stopped and only the logic for monitoring RESET, HOLD, and FLUSH# remains active. With SUSPA# asserted, the CLK2 input to the microprocessor can be stopped in either phase. Stopping the CLK2 input further reduces current required by the microprocessor.

To resume operation, restart the CLK2 input (if stopped) and negate the SUSP# input. The TI486SXLC2 processors can enter clock-doubled mode (subsection 3.2.1.1, *Entering Clock-Doubled Mode*) once the CLK2 input reaches the desired frequency within the PLL lock range. The processor then resumes instruction fetching and begins execution in the instruction stream at the point where it 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 using the SUSP bit in the CCR0 Configuration register.

3.2.2.2 Suspend Acknowledge (SUSPA#)

The suspend acknowledge (SUSPA#) output indicates that the TI486SXLC series microprocessor has entered the suspend mode as a result of SUSP# assertion or execution of a HALT instruction. If SUSPA# is asserted and the CLK2 input is switching, the microprocessor continues to recognize FLT#, RESET, HOLD, and FLUSH#. In addition, the TI486SXLC2 microprocessor may stay in clock-doubled mode while the CLK2 input is switching. If suspend mode was entered as the result of a HALT instruction, the microprocessor also continues to monitor the NMI input and the unmasked INTR input. Detection of INTR or NMI forces the microprocessor to exit suspend mode and begin execution of the appropriate interrupt service routine. The CLK2 input to the processor can be stopped after SUSPA# has been asserted to reduce the power requirement of the microprocessor further. For this case, the TI486SXLC2 microprocessor must be brought out of clock-doubled mode before stopping the CLK2 input to prevent a synchronization error. The SUSPA# output is disabled (floated) following reset and can be enabled using the SUSP bit in the CCR0 Configuration register.

Table 3–5 shows the state of the TI486SXLC series microprocessor signals when the device is in suspend mode.

Table 3–5. TI486SXLC Signal States During Suspend Mode

Signal Name	Signal State During Hold Acknowledge	Signal State During Halt-Initiated Suspend Mode
A20M#	Ignored	Ignored
A23–A1	1	1
ADS#	1	1
BHE#, BLE#	0	0
BUSY#	Ignored	Ignored
D15–D0	Float	Float
D/C#	1	1
ERROR#	Ignored	Ignored
FLT#	Input recognized	Input recognized
FLUSH#	Input recognized	Input recognized
HLDA	0	0
HOLD	Input recognized	Input recognized
INTR	Latched	Input recognized
KEN#	Ignored	Ignored
LOCK#	1	1
M/IO#	0	0
NA#	Ignored	Ignored
NMI	Latched	Input recognized
PEREQ	Ignored	Ignored
READY#	Ignored	Ignored
RESET	Input recognized	Input recognized
SMADS#	1	1
SMI#	Latched	Input recognized
SUSP#	Input recognized	Ignored
SUSPA#	0	0
W/R#	0	0

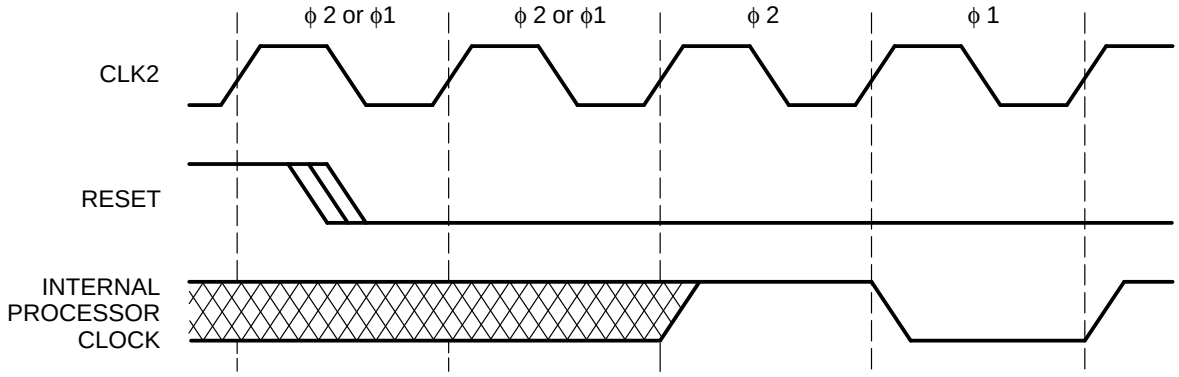
3.3 Reset Timing and Internal Clock Synchronization

RESET is the highest priority input signal and interrupts any processor activity when it is asserted. When RESET is asserted, the microprocessor aborts any bus cycle. Idle, hold-acknowledge, and suspend states are also discontinued, and the reset state is established. RESET is used when the microprocessor is powered up to initialize the CPU to a known valid state and to synchronize the internal CPU clock with external clocks. The TI486SXLC2 microprocessors are initialized to nonclock-doubled mode when RESET goes active.

RESET must be asserted for at least 15 CLK2 periods to ensure recognition by the microprocessor. If the self-test feature is invoked, RESET must be asserted for at least 80 CLK2 periods. RESET pulses of less than 15 CLK2 periods may not have sufficient time to propagate throughout the microprocessor and may not be recognized. RESET pulses of less than 80 CLK2 periods followed by a self-test request may incorrectly report a self-test failure when none has occurred .

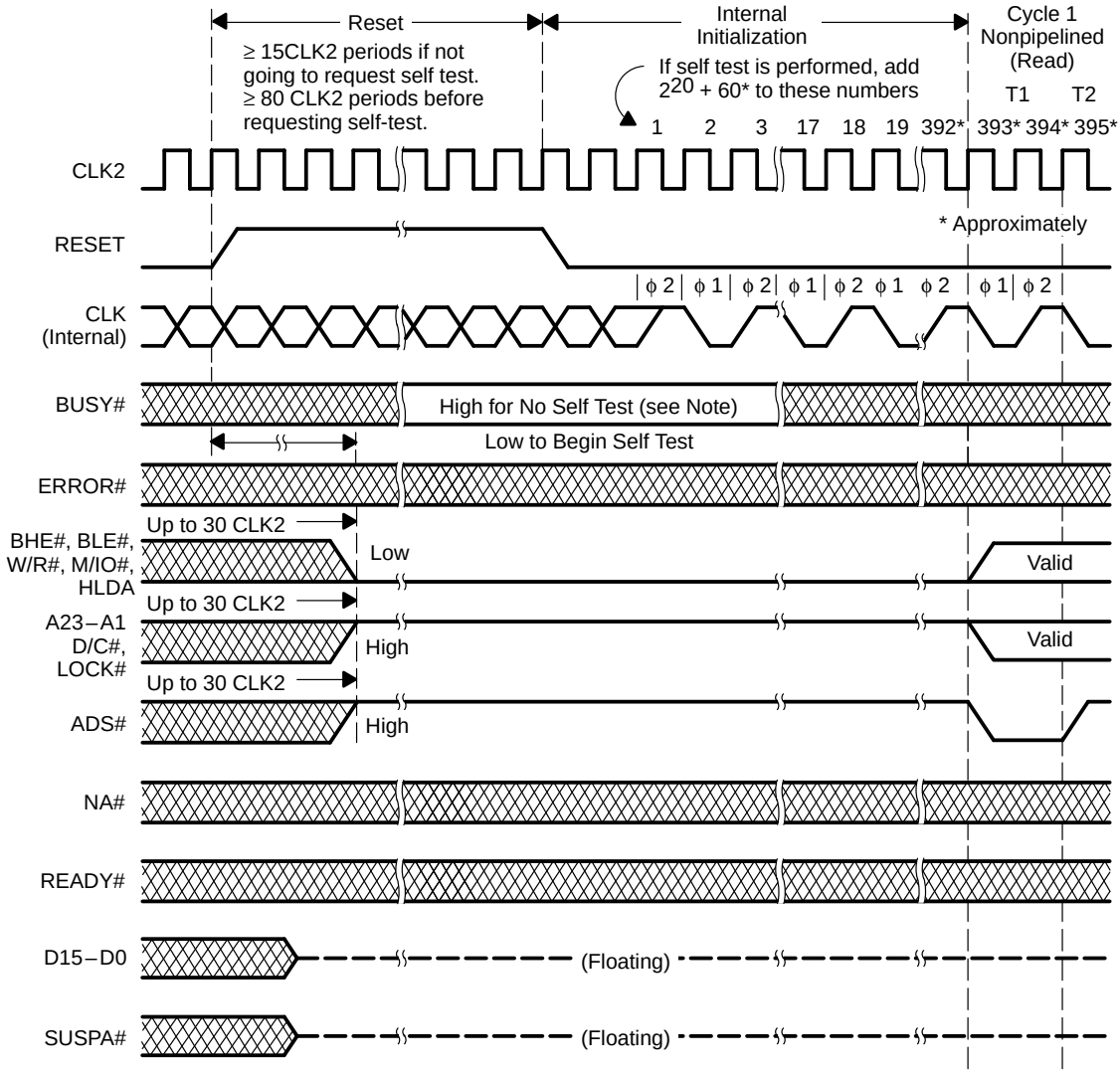
If the RESET falling edge meets specified setup and hold times, the internal processor clock phase is synchronized as illustrated in Figure 3–2. The TI486SXLC internal processor clock is half the frequency of the CLK2 input and each CLK2 cycle corresponds to an internal CPU clock phase (ϕ). Phase two ($\phi 2$) of the internal clock is defined as the second rising edge of CLK2 following the falling edge of RESET. The TI486SXLC2 internal core clock is the same frequency as the CLK2 input, and the internal bus interface clock is half the frequency of the CLK2 input. Phase two of the internal clock is defined as the second rising edge of CLK2 following the falling edge of RESET.

Figure 3–2. TI486SXLC Internal Processor Clock Synchronization



Following the falling edge of RESET (and after self test if it was requested), the microprocessor performs an internal initialization sequence for approximately 400 CLK2 periods. The microprocessor self-test feature is invoked if the BUSY# input is in the active (low) state when RESET falls inactive. The self-test sequence requires approximately (2^{20} + 60) CLK2 periods to complete. Even if the self test indicates a problem, the microprocessor attempts to proceed with the reset sequence. Figure 3–3 illustrates the bus activity and timing during the microprocessor reset sequence.

Figure 3–3. TI486SXLC Bus Activity From RESET Until First Code Fetch



Note: BUSY# should be held stable for 80 CLK2 periods before and after the CLK2 period in which RESET falling edge occurs.

Upon completion of self-test, the EAX register contains 0000 0000h if the microprocessor passed its internal self test with no problems. Any nonzero value in the EAX register indicates that the microprocessor is faulty.

3.4 Bus Operation and Functional Timing

The TI486SXLC series microprocessor communicates with the external system through separate, parallel buses for data and address. This is commonly called a demultiplexed address/data bus. This demultiplexed bus eliminates the need for address latches required in multiplexed address/data bus configurations, where the address and data are presented on the same pins at different times.

TI486SXLC series microprocessor instructions can act on memory data operands consisting of 8-bit bytes, 16-bit words, or 32-bit double words. The microprocessor bus architecture allows for bus transfers of these operands without restrictions on physical address alignment. Any byte boundary may require more than one bus cycle to transfer the operand. This feature is transparent to the programmer.

The microprocessor data bus (D15–D0) is a 16-bit-wide bidirectional bus. The microprocessor drives the data bus during write bus cycles, and the external system hardware drives the data bus during read bus cycles. The address bus provides a 24-bit value. Twenty-three signals for the 23 upper-order address bits (A23–A1) define which 16-bit word is being accessed. Two byte-enable signals (BHE# and BLE#) directly indicate which of the two bytes within the word is active.

Every bus cycle begins with assertion of the address strobe (ADS#). ADS# indicates that the microprocessor has issued a new address and new bus-cycle-definition signals. A bus cycle is defined by four signals: M/IO#, W/R#, D/C#, and LOCK#. M/IO# defines whether a memory or I/O operation is occurring, W/R# defines the cycle as read or write, and D/C# indicates whether a data or control cycle is in effect. LOCK# indicates that the current cycle is a locked bus cycle. Every bus cycle completes when the system hardware returns READY# asserted.

The TI486SXLC series microprocessor performs the following bus-cycle types:

- Memory read
- Locked memory read
- Memory write
- Locked memory write
- I/O read (or coprocessor read)
- I/O write (or coprocessor write)
- Interrupt acknowledge (always locked)
- Halt/shutdown

When the microprocessor has no pending bus requests, the bus enters the idle state. There is no encoding of the idle state on the bus-cycle-definition signals; however, the idle state can be identified by the absence of further assertions of ADS# following a completed bus cycle.

Note that all bus diagrams apply to all TI486SXLC series microprocessors. The TI486SXLC2 clock-doubled feature does not change the external microprocessor bus interface.

3.4.1 Bus Cycles Using Nonpipelined Addressing

The shortest time unit of bus activity is a bus state, commonly called a T state. A bus state is one internal processor clock period in duration (two CLK2 periods in nonclock-doubled mode and one CLK2 period in clock-doubled mode). A complete data transfer occurs during a bus cycle, composed of two or more bus states.

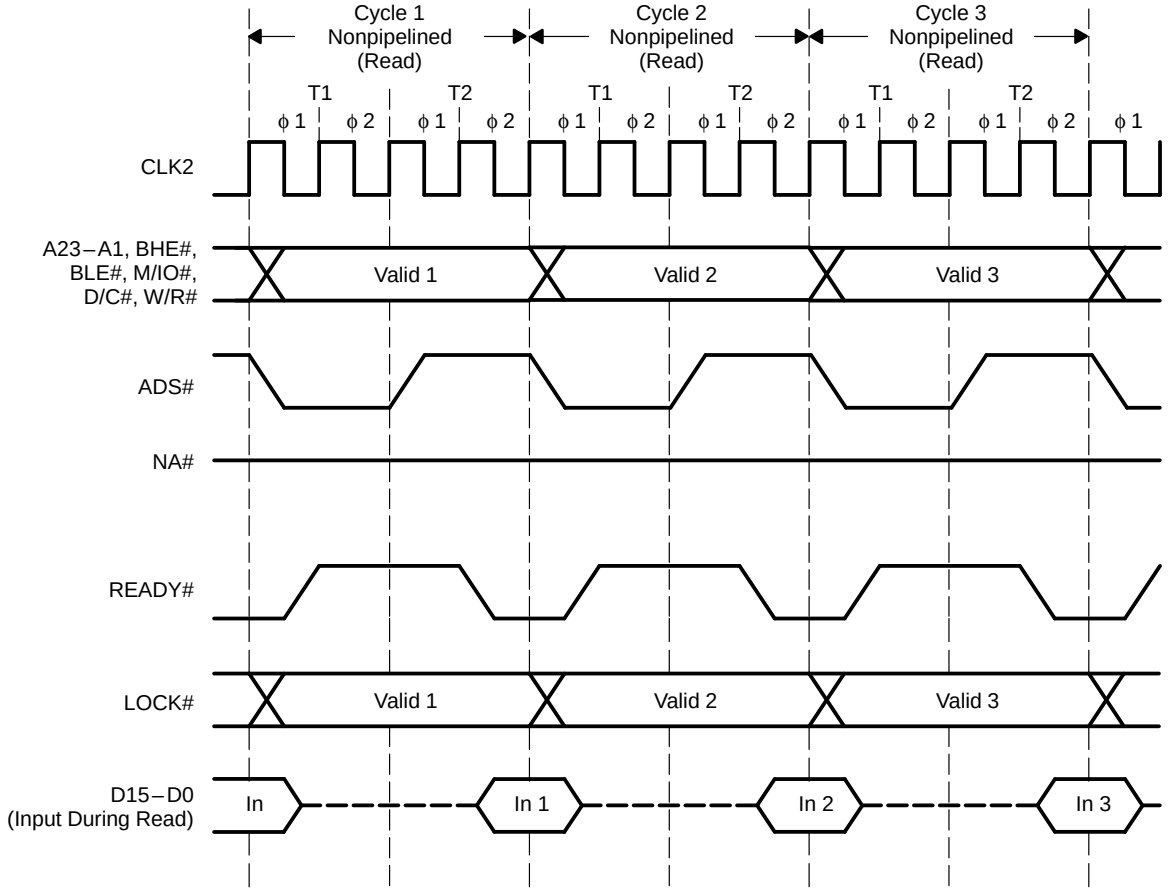
3.4.1.1 Nonpipelined Bus States

The first state of a nonpipelined bus cycle is called T1. During phase one (ϕ1, first CLK2) of T1, the address bus and bus-cycle-definition signals are driven valid and, to signal their availability, address strobe (ADS#) is simultaneously asserted.

The second bus state of a nonpipelined cycle is called T2. T2 terminates a bus cycle with the assertion of the READY# input, and valid data is either read or written depending on the bus-cycle type. The fastest microprocessor bus cycle requires only these two bus states. READY# is ignored at the end of the T1 state.

Three consecutive bus read cycles, each consisting of two bus states, are shown in Figure 3–4.

Figure 3–4. TI486SXLC Fastest Nonpipelined Read Cycles



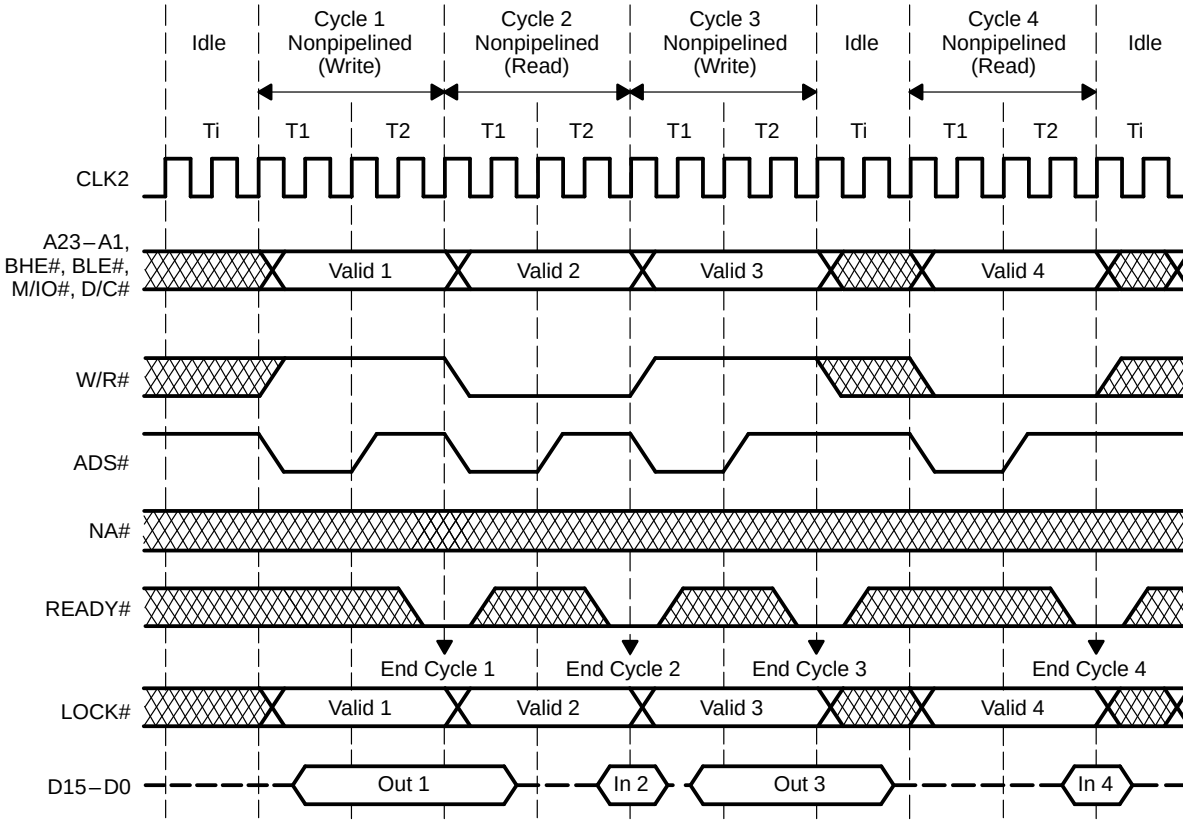
Note: Fastest nonpipelined bus cycles consist of T1 and T2.

3.4.1.2 Nonpipelined Read and Write Cycles

Any bus cycle can be performed with nonpipelined address timing. Figure 3–5 shows a mixture of read and write cycles with nonpipelined address timing. When a read cycle is performed, the microprocessor floats its data bus, and the externally addressed device then drives the data. The microprocessor requires that all data-bus pins be driven to a valid logic state (high or low) at the end of each read cycle, when READY# is asserted. When a read cycle is acknowledged by READY# asserted in the T2 bus state, the microprocessor latches the information present at its data-bus pins and terminates the cycle.

When a write cycle is performed, the data bus is driven by the microprocessor beginning in phase two of T1. When a write cycle is acknowledged, the write data remains valid throughout phase one of the next bus state to provide write-data hold time.

Figure 3–5. TI486SXLC Various Nonpipelined Bus Cycles (No Wait States)



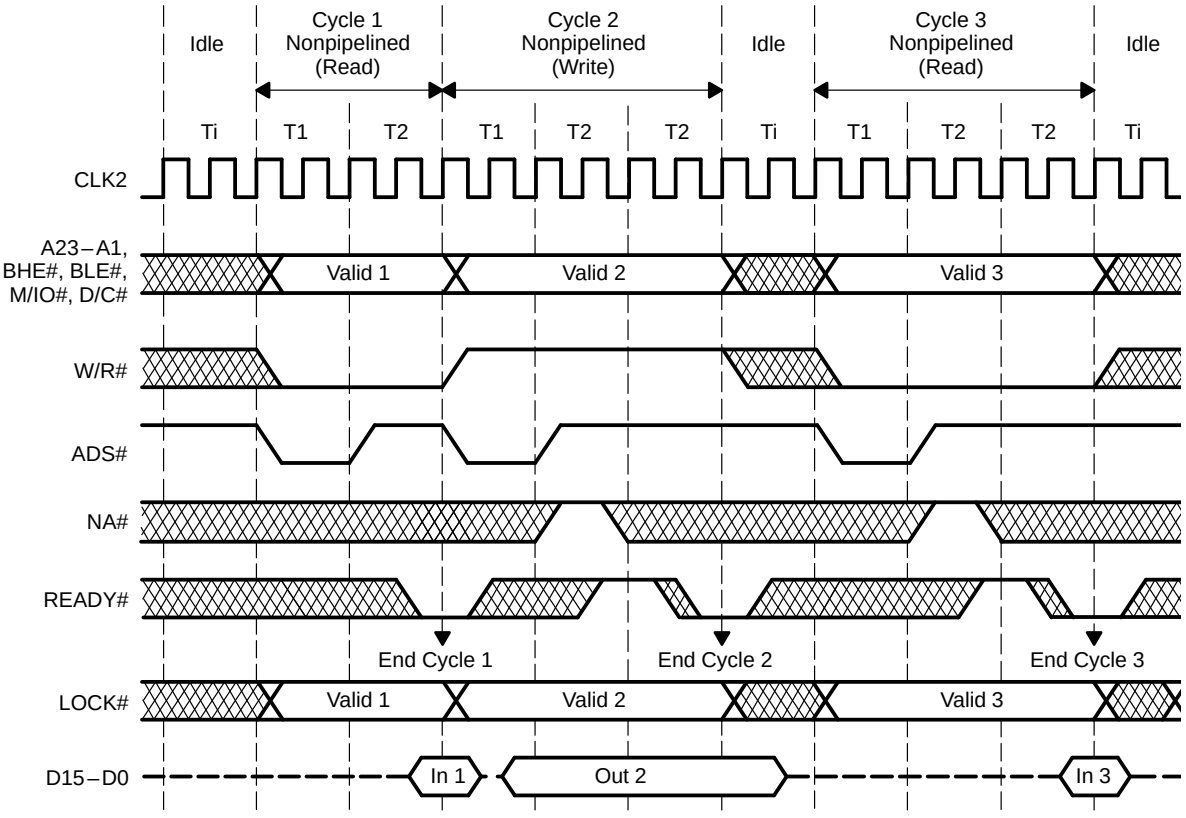
Note: Idle states are introduced arbitrarily.

3.4.1.3 Nonpipelined Wait States

Once a bus cycle begins, it continues until acknowledged by the external system hardware using the READY# input. Acknowledging the bus cycle at the end of the first T2 results in the shortest possible bus cycle, requiring only T1 and T2. However, If READY# is not immediately asserted, T2 states are repeated indefinitely until the READY# input is sampled active. These intermediate T2 states are referred to as wait states. If the external system hardware is not able to receive or deliver data in two bus states, READY# is withheld and adds at least one wait state to the bus cycle. Thus, on an address-by-address basis, the system is able to define how fast a bus cycle completes.

Figure 3–6 illustrates nonpipelined bus cycles with one wait state added to cycles 2 and 3. READY# is sampled inactive at the end of the first T2 state in cycles 2 and 3. Therefore, the T2 state is repeated until READY# is sampled active at the end of the second T2, and the cycle is then terminated. The microprocessor ignores the READY# input at the end of the T1 state.

Figure 3–6. TI486SXLC Various Nonpipelined Bus Cycles With Different Numbers of Wait States

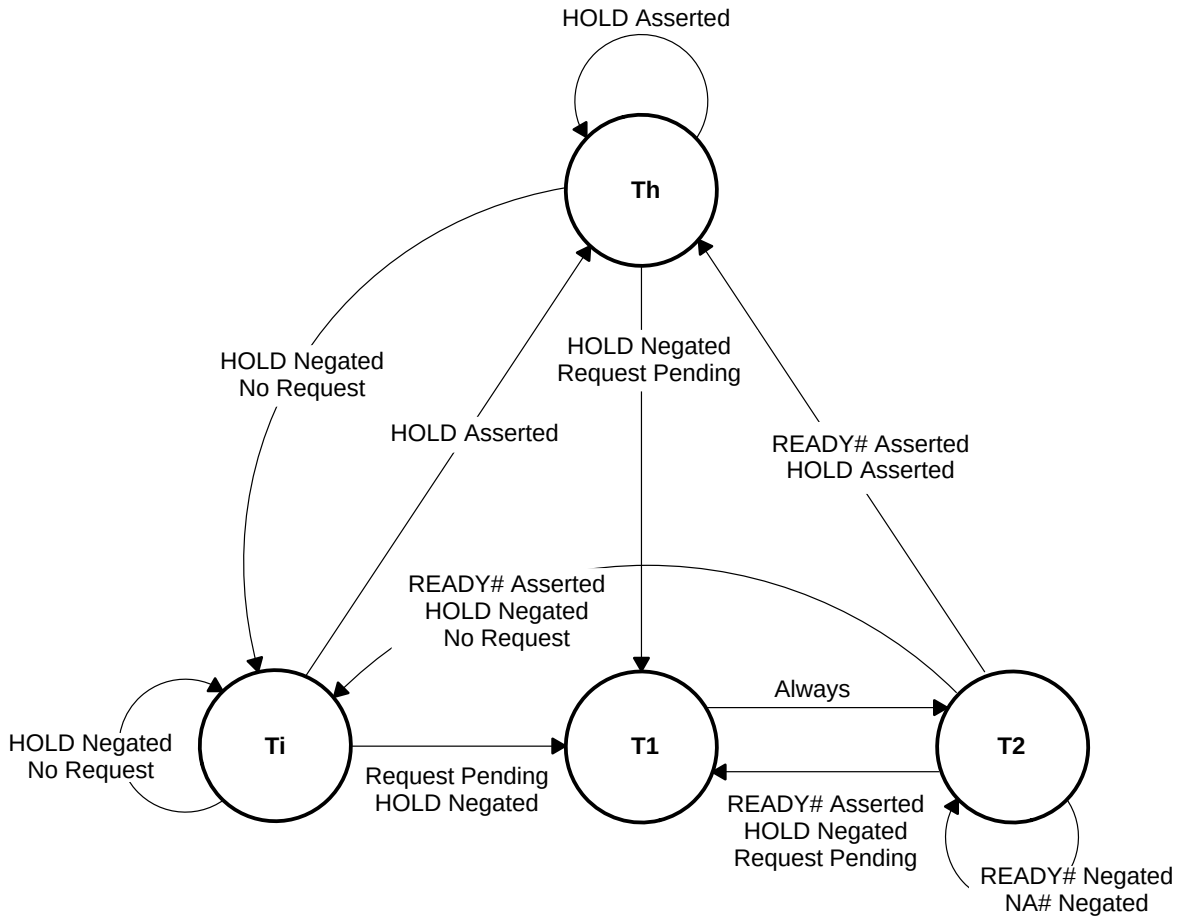


Note: Idle states are introduced arbitrarily.

3.4.1.4 Initiating and Maintaining Nonpipelined Cycles

The bus states and transitions for nonpipelined addressing are illustrated in Figure 3–7. The bus can switch between four possible states: T1, T2, Ti, and Th. Active bus cycles consist of T1 and T2 states, with T2 repeated for wait states. Bus cycles always begin with a single T1 state. T1 is always followed by a T2 state. If a bus cycle is not acknowledged during a given T2 and NA# is inactive, T2 repeats, resulting in a wait state. When a cycle is acknowledged during T2, the following state is T1 of the next bus cycle when a bus request is pending internally. If no internal bus request is pending, the Ti state is entered. If the HOLD input is asserted and the microprocessor is ready to enter the hold-acknowledge state, the Th state is entered.

Figure 3–7. TI486SXLC Nonpipelined Bus States



Bus States:
T1 – First clock of a nonpipelined bus cycle (CPU drives new address and asserts ADS#)
T2 – Subsequent clocks of a bus cycle when NA# has not been sampled asserted in the current bus cycle
Ti – Idle state
Th – Hold acknowledge (CPU asserts HLDA)

The fastest bus cycle consists of two states: T1 and T2.

Due to the demultiplexed bus, address pipelining gives the external hardware an additional T state of access time without inserting a wait state. The processor always uses nonpipelined address timing after the reset sequence and following any idle bus state. Pipelined or nonpipelined address timing is then determined on a cycle-by-cycle basis using the NA# input. When address pipelining is not used, the address and bus-cycle definition remain valid during all wait states. When wait states are added and nonpipelined address timing is necessary, NA# should be negated during each T2 state of the bus cycle except the last one.

3.4.2 Bus Cycles Using Pipelined Addressing

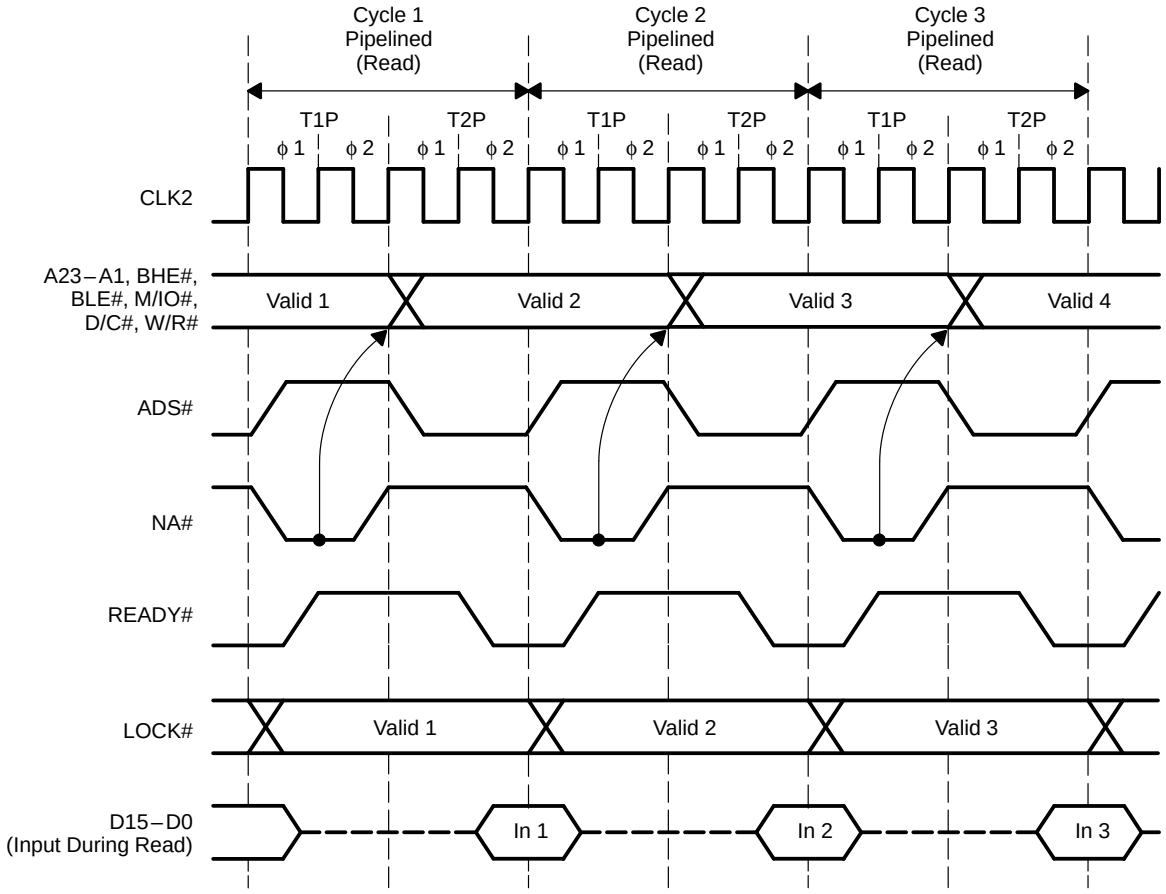
Address pipelining lets the system request the address and bus-cycle definition of the next internally pending bus cycle before the current bus cycle is acknowledged with READY# asserted. If address pipelining is used, the external system hardware has an extra T state of access time to transfer data. Address pipelining is controlled cycle-by-cycle by the state of the NA# input.

3.4.2.1 Pipelined Bus States

Pipelined addressing is always initiated by asserting NA# during a nonpipelined bus cycle. Within the nonpipelined bus cycle, NA# is sampled at the beginning of phase two of each T2 state and is only acknowledged by the microprocessor during wait states. When address pipelining is acknowledged, the address (BHE#, BLE#, and A23–A1) and bus-cycle definition (W/R#, D/C#, and M/IO#) of the next bus cycle are driven before the end of the nonpipelined cycle. The address status output (ADS#) is asserted simultaneously to indicate validity of these signals. Once in effect, address pipelining is maintained in successive bus cycles by continuing to assert NA# during the pipelined bus cycles.

As in nonpipelined bus cycles, the fastest bus cycles using a pipelined address require only two bus states. Figure 3–8 illustrates the fastest read cycles using pipelined address timing. The two bus states for pipelined addressing are T1P and T2P or T1P and T2I. The T1P state is entered following completion of the bus cycle in which the pipelined address and bus-cycle-definition information was made available, and it is the first bus state of every pipelined bus cycle. In other words, the T1P state follows a T2 state if the previous cycle was nonpipelined, and follows a T2P state if the previous cycle was pipelined.

Figure 3–8. TI486SXLC Fastest Pipelined Read Cycles



Note: Fastest pipelined bus cycles consist of T1P and T2P.

Within the pipelined bus cycle, NA# is sampled at the beginning of phase two ($\phi 2$) of the T1P state. If the microprocessor has an internally pending bus request and NA# is asserted, the T1P state is followed by a T2P state and the address and bus-cycle definition for the next pending bus request is made available. If no pending bus request exists, the T1P state is followed by a T2I state regardless of the state of NA# and no new address or bus-cycle information is driven.

The pipelined bus cycle is terminated in either the T2P or T2I states with the assertion of the READY# input, and valid data is either input or output depending on the bus cycle type. READY# is ignored at the end of the T1P state.

3.4.2.2 Pipelined Read and Write Cycles

Any bus cycle can be performed with pipelined address timing. When a read cycle is performed, the microprocessor floats its data bus and the externally addressed device drives the data. When a read cycle is acknowledged by READY# asserted in either the T2P or T2I bus state, the microprocessor latches the information present at its data pins and terminates the cycle.

When a write cycle is performed, the data bus is driven by the microprocessor beginning in phase two ($\phi 2$) of T1P. When a write cycle is acknowledged, the

write data remains valid throughout phase one ($\phi 1$) of the next bus state to provide write-data hold time.

3.4.2.3 Pipelined Wait States

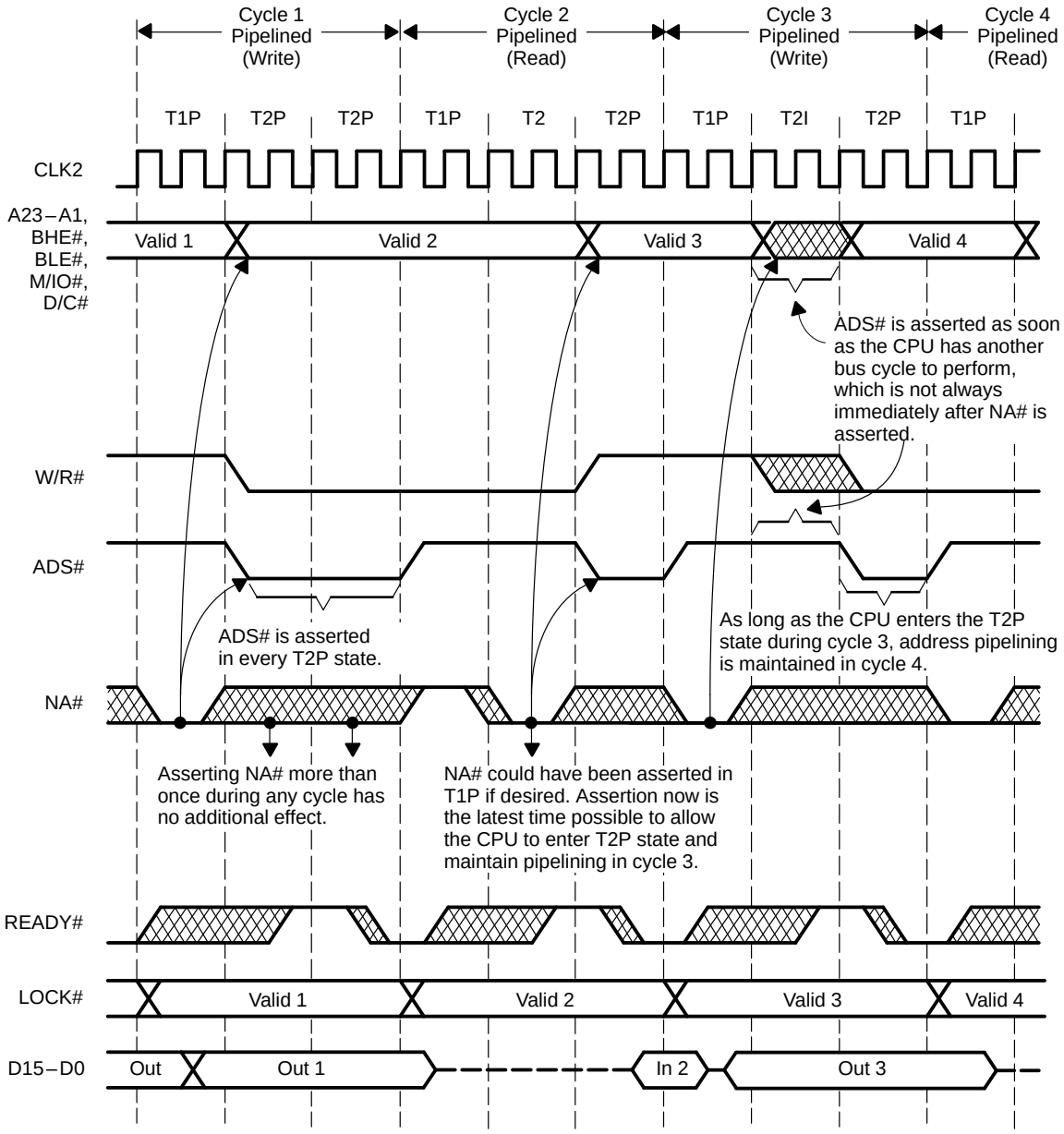
Once a pipelined bus cycle begins, it continues until acknowledged by the external system hardware using the microprocessor READY# input. Acknowledging the bus cycle at the end of the first T2P or T2I state results in the shortest possible pipelined bus cycle. If READY# is not immediately asserted, however, T2P or T2I states are repeated indefinitely until the READY# input is sampled active. Additional T2P or T2I states are referred to as wait states.

Figure 3–9 illustrates pipelined bus cycles with one wait state added to cycles 1 through 3. Cycle 1 is a pipelined cycle with NA# asserted during T1P and a pending bus request. READY# is sampled inactive at the end of the first T2P state in cycle 1. Therefore, the T2P state is repeated until READY# is sampled active at the end of the second T2P, and the cycle is then terminated. The microprocessor ignores the READY# input at the end of the T1P state. ADS#, the address, and the bus-cycle-definition signals for the pending bus cycle are all valid during each of the T2P states. Also, asserting NA more than once during the cycle has no additional effect. Pipelined addressing can only output information for the next bus cycle.

Cycle 2 in Figure 3–9 illustrates a pipelined cycle, with one wait state, where NA# is not asserted until the second bus state in the cycle. In this case, the CPU enters the T2 state following T1P because NA# is not asserted. During the T2 state the microprocessor samples NA# asserted. Because a bus request is pending internally and READY# is not active, the CPU enters the T2P state and asserts ADS#, a valid address, and bus-cycle-definition information for the pending bus cycle. The cycle is then terminated by an active READY# at the end of the T2P state.

Cycle 3 of Figure 3–9 illustrates the case where no internal bus request exists until the last state of a pipelined cycle with wait states. In cycle 3, NA# is asserted in T1P, requesting the next address. Because the CPU does not have an internal bus request pending, the T2I state is entered. However, by the end of the T2I state, a bus request exists. Because READY# is not asserted, a wait state is added. The CPU then enters the T2P state and asserts ADS#, a valid address, and bus-cycle-definition information for the pending bus cycle. As long as the CPU enters the T2P state at some point during the bus cycle, pipelined addressing is maintained. NA# needs to be asserted only once during the bus cycle to request pipelined addressing.

Figure 3–9. TI486SXLC Various Pipelined Cycles (One Wait State)



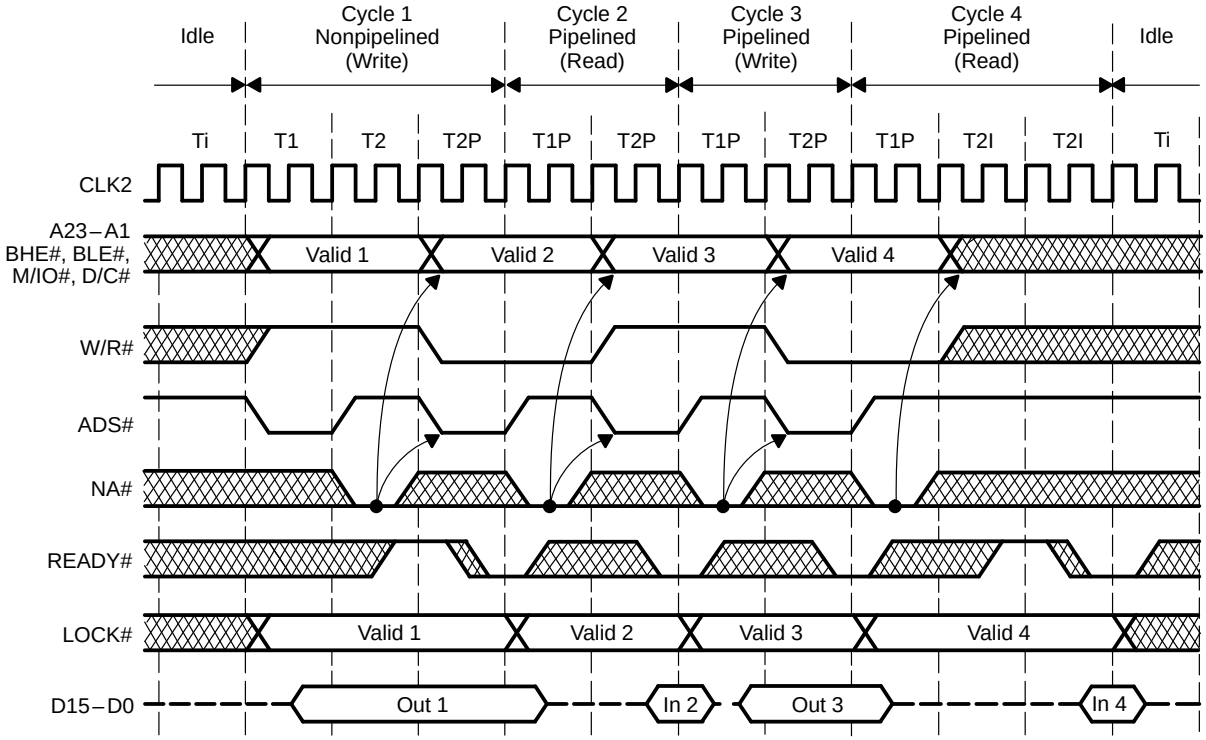
3.4.2.4 Initiating and Maintaining Pipelined Cycles

Pipelined addressing is always initiated by asserting NA# during a nonpipelined bus cycle with at least one wait state. The first bus cycle following reset, an idle bus, or a hold-acknowledge state is always nonpipelined. Therefore, the microprocessor always issues at least one nonpipelined bus cycle following reset, idle, or hold acknowledge before pipelined addressing takes effect.

Once a bus cycle is in progress and the current address has been valid for one entire bus state, the NA# input is sampled at the end of every phase one until the bus cycle is acknowledged. Once NA# is sampled active, the microprocessor is free to drive a new address and bus-cycle definition on the bus as early as the next bus state and as late as the last bus state in the cycle.

Figure 3–10 illustrates the fastest transition possible to pipelined addressing following an idle bus state. In cycle 1, NA# is driven during state T2. Thus, cycle 1 makes the transition to pipelined address timing, since it begins with T1 but ends with T2P. Because the address for cycle 2 is available before cycle 2 begins, cycle 2 is called a pipelined bus cycle, and it begins with a T1P state. Cycle 2 begins as soon as READY# assertion terminates cycle 1.

Figure 3–10. TI486SXLC Fastest Transition to Pipelined Address Following Bus-Idle State

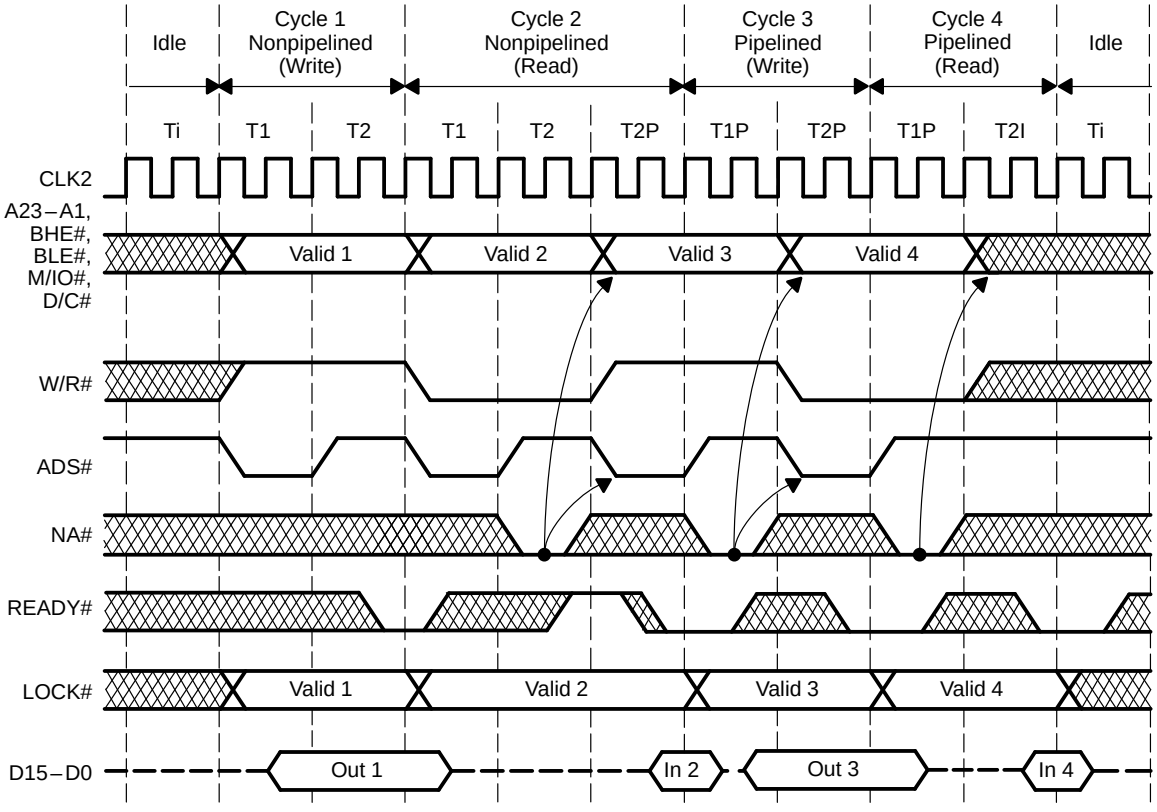


Note: Following any idle bus state (Ti), the address is always nonpipelined and NA# is sampled only during wait states. To start address pipelining after an idle state, a nonpipelined cycle with at least one wait state (cycle 1 above) is required. The pipelined cycles (2, 3, and 4 above) are shown with various numbers of wait states.

Figure 3–11 illustrates the transition to pipelined addressing during a burst of bus cycles. Cycle 2 makes the transition to pipelined addressing. Comparing cycle 2 to cycle 1 of Figure 3–10 (on page 3-28) illustrates that a transition cycle is the same when it occurs and consists of at least T1, T2 (NA# is asserted at that time), and T2P (provided the microprocessor has an internal bus request already pending). T2P states are repeated if wait states are added to the cycle. Cycles 2, 3, and 4 in Figure 3–11 show that once address pipelining is achieved, it can be maintained with two-state bus cycles consisting only of T1P and T2P.

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting NA# and detecting that the microprocessor enters T2P during the current bus cycle. The current bus cycle must end in state T2P for pipelining to be maintained in the next cycle. T2P is identified by the assertion of ADS#. Figure 3–10 and Figure 3–11 each show pipelining ending after cycle 4. This occurs because the microprocessor does not have an internal bus request prior to the acknowledgment of cycle 4.

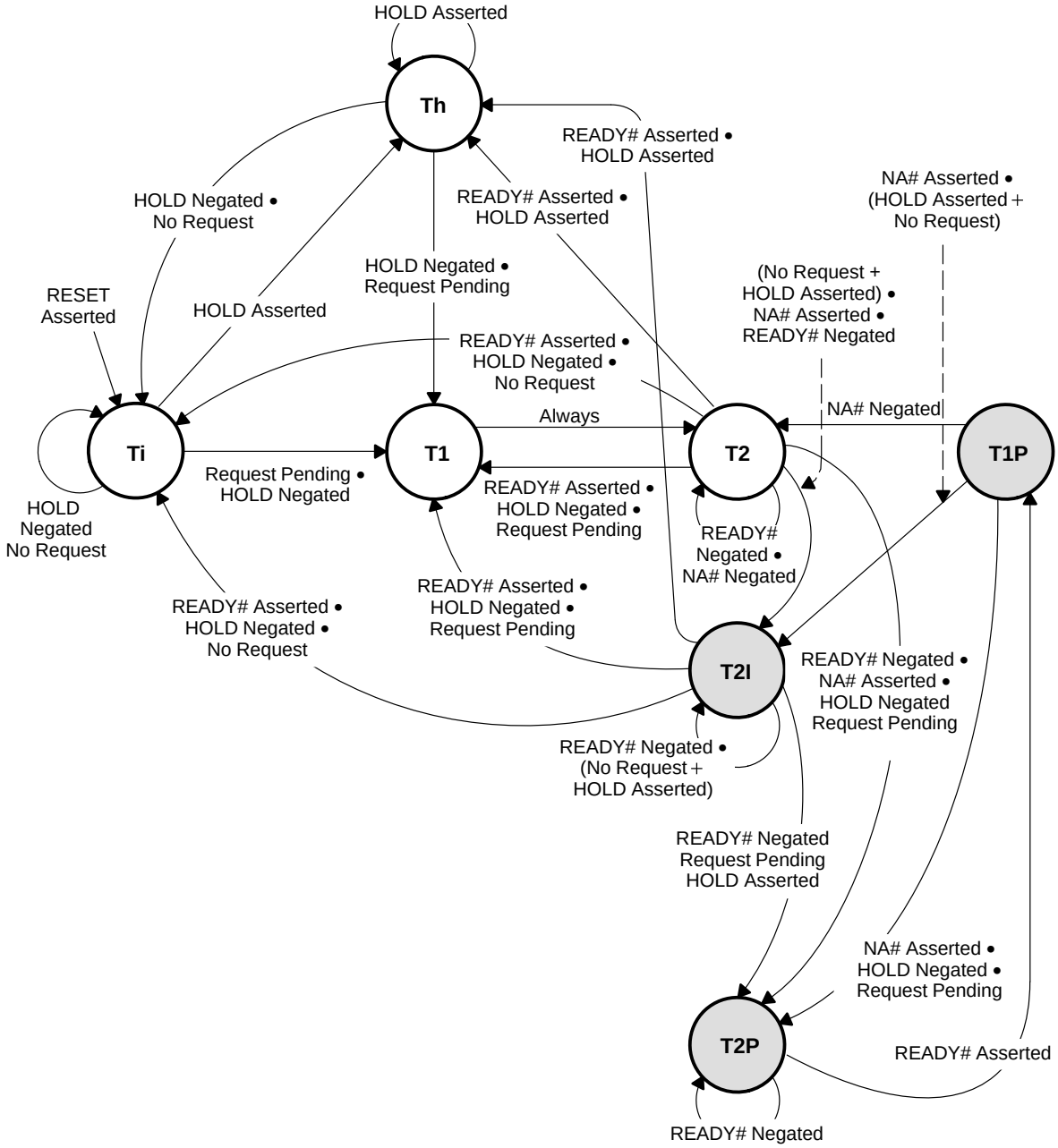
Figure 3–11. TI486SXLC Transition to Pipelined Address During Burst of Bus Cycles



Note: Following any idle bus state (Ti), addresses are nonpipelined bus cycles, and NA# is sampled only during wait states. Therefore, to begin address pipelining during a group of nonpipelined bus cycles requires a nonpipelined cycle with at least one wait state (cycle 2 above).

The complete bus-state-transition diagram, including operation with pipelined address, is given in Figure 3–12. This is a superset of the diagram for nonpipelined address. The three additional bus states for pipelined address are shaded.

Figure 3–12. TI486SXLC Complete Bus States



Bus States:

T1 – First clock of a nonpipelined bus cycle (CPU drives new address and asserts ADS#)
T2 – Subsequent clocks of a bus cycle when NA# has not been sampled asserted in the current bus cycle
T2I – Subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle but there is not yet an internal bus request pending (CPU does not drive a new address or assert ADS#)
T2P – Subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle and there is an internal bus request pending (CPU drives new address and asserts ADS#)
T1P – First clock of a pipelined bus cycle
Ti – Idle state
Th – Hold acknowledge state (CPU asserts HLDA)

3.4.3 Locked Bus Cycles

When the LOCK# signal is asserted, the TI486SXLC series microprocessors do not allow other bus master devices to gain control of the system bus. LOCK# is driven active in response to executing certain instructions with the LOCK prefix. The LOCK prefix allows indivisible read/modify/write operations on memory operands. LOCK# is also active during interrupt-acknowledge cycles.

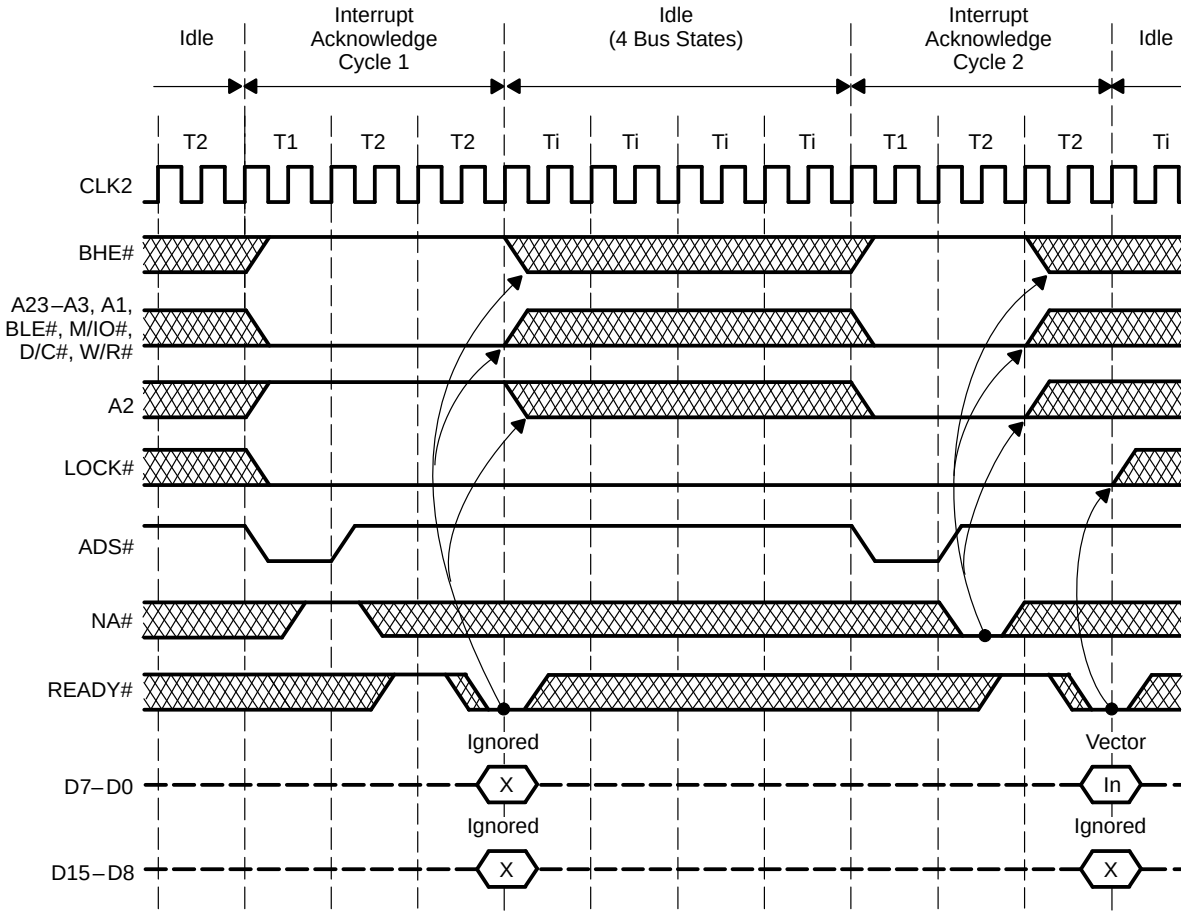
LOCK# is activated on the CLK2 edge that begins the first locked bus cycle and is deactivated when READY# is returned at the end of the last locked bus cycle. When the microprocessor is using nonpipelined addressing, LOCK# is asserted during phase one ($\phi 1$) of T1. When it is using pipelined addressing, LOCK# is driven valid during phase one of T1P.

Figure 3–4 through Figure 3–6 on pages 3-20 through 3-22 illustrate LOCK# timing during nonpipelined cycles. Figure 3–8 through Figure 3–11 on pages 3-25 through 3-29 cover the pipelined-address case.

3.4.4 Interrupt-Acknowledge Cycles

The TI486SXLC series microprocessors are interrupted by an external source via an input request on the INTR input (when interrupts are enabled). The microprocessor responds with two locked interrupt-acknowledge cycles. These bus cycles are similar to read cycles. Each cycle is terminated by READY# sampled active as shown in Figure 3–13.

Figure 3–13. TI486SXLC Interrupt-Acknowledge Cycles



Note: Interrupt vector (0–255) is read on D7–D0 at the end of the second interrupt-acknowledge bus cycle. Because each interrupt- acknowledge bus cycle is followed by idle bus states, asserting NA# has no practical effect.

The state of the A2 pin distinguishes the first and second interrupt-acknowledge cycles. The address driven during the first interrupt-acknowledge cycle is 4h (A23–A3, A1, BLE#=0; A2, BHE#=1). The address driven during the second interrupt-acknowledge cycle is 0h (A23–A1, BLE#=0; BHE#=1).

To assure that the interrupt-acknowledge cycles are executed indivisibly, the LOCK# output is asserted from the beginning of the first interrupt-acknowledge cycle until the end of the second interrupt-acknowledge cycle. In clock-doubled mode, four idle bus states (Ti) are inserted by the microprocessor between the two interrupt-acknowledge cycles. In nonclock-doubled mode, eight idle bus states are inserted.

The interrupt vector is read at the end of the second interrupt cycle. The microprocessor reads the vector from D7–D0 of the data bus. The vector indicates the specific interrupt number (from 0–255) requiring service. Throughout the balance of the two interrupt cycles, D15–D0 float. At the end of the first interrupt-acknowledge cycle, any data presented to the microprocessor is ignored.

3.4.5 Halt and Shutdown Cycles

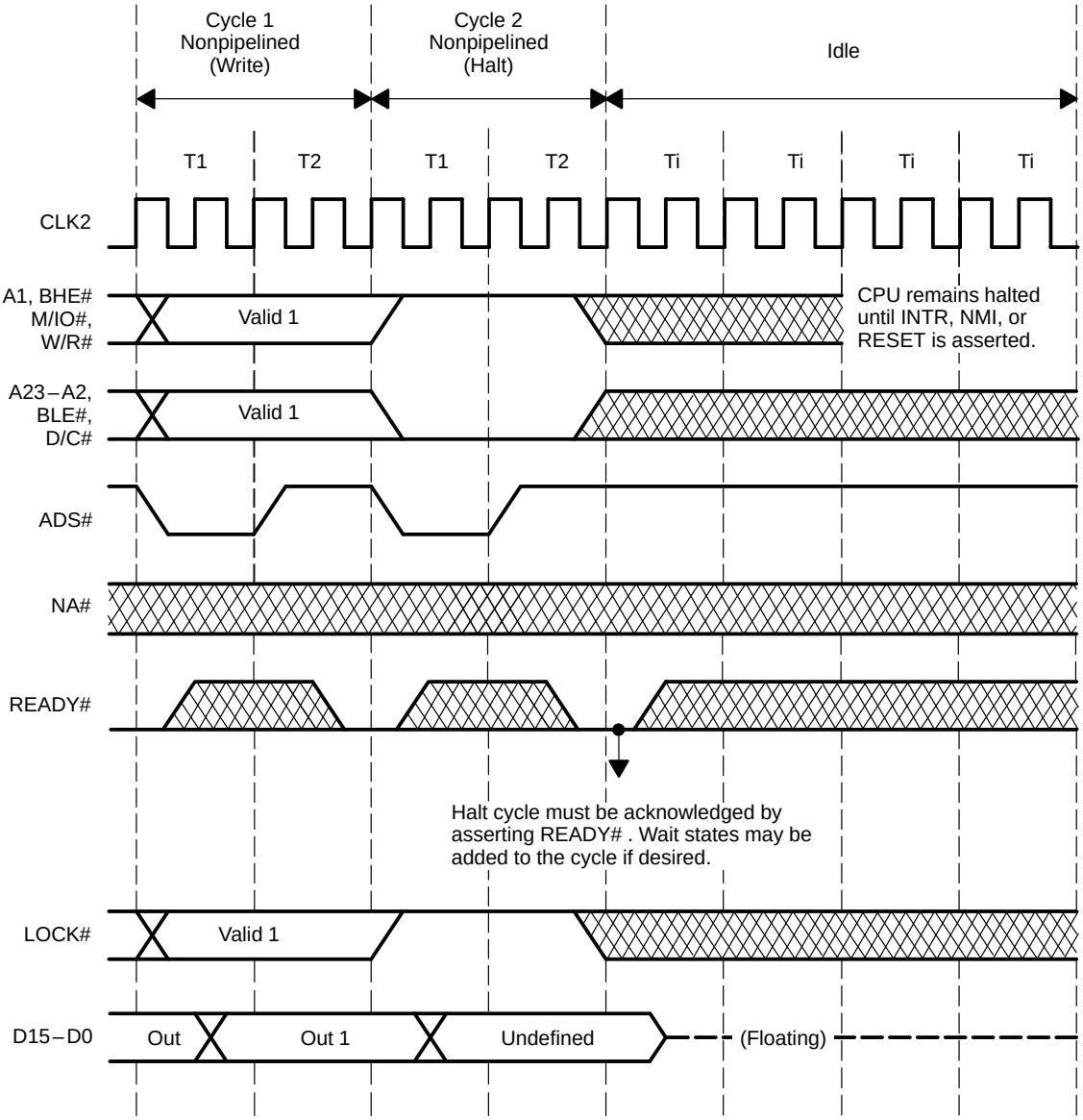
Executing the HLT instruction or detecting a severe error causes the microprocessor to halt operation or shutdown further processing. The microprocessor signals a halt or shutdown through a halt- or shutdown-indication cycle, respectively.

3.4.5.1 Halt Indication Cycle

Executing the HLT instruction causes the microprocessor execution unit to cease operation. The microprocessor signals its entrance into the halt state by performing a halt indication cycle. The halt indication cycle is identified by the state of the bus-cycle-definition signals (M/IO#=1, D/C#=0, W/R#=1, and LOCK#=1) and an address of 2h (A23–A2=0, A1=1, BHE#=1, and BLE#=0).

The halt indication cycle must be acknowledged by asserting READY#. A halted microprocessor resumes execution when INTR (if interrupts are enabled), NMI, SMI#, or RESET is asserted. Figure 3–14 illustrates a nonpipelined halt cycle.

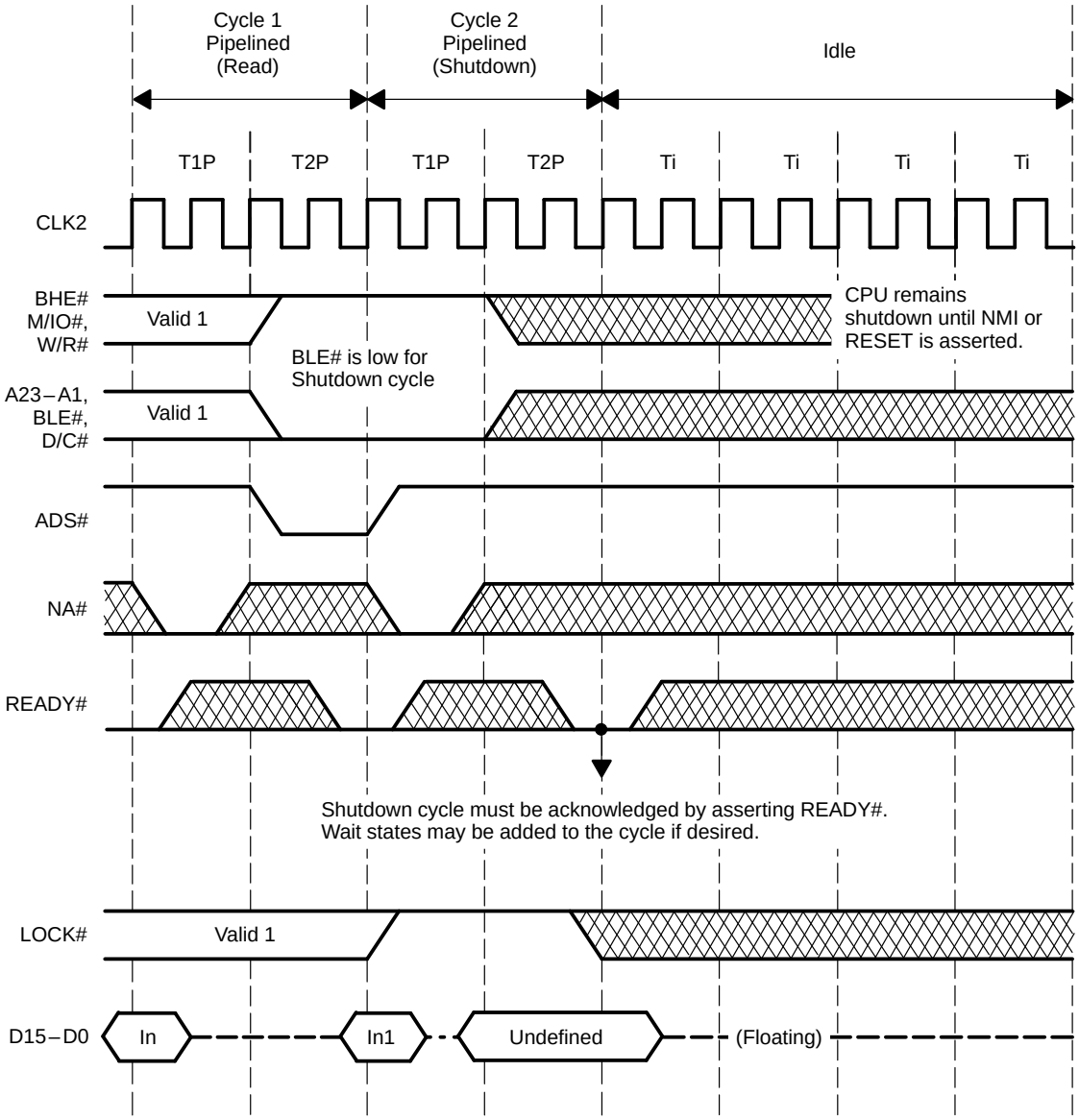
Figure 3–14. TI486SXLC Nonpipelined Halt Cycle



3.4.5.2 Shutdown Indication Cycle

Shutdown occurs when the microprocessor detects a severe error that prevents further processing. The TI486SXLC series microprocessor shuts down as a result of a protection fault while attempting to process a double fault as well as the conditions referenced in Chapter 2, *Programming Interface*. A shutdown indication cycle is performed, which signals the microprocessor's entrance into the shutdown state. The shutdown indication cycle is identified by the state of the bus-cycle-definition signals (M/IO#=1, D/C#=0, W/R#=1, and LOCK#=1) and an address of 0h (A23–A1=0, BHE#=1, and BLE#=0). The shutdown indication cycle must be acknowledged by asserting READY#. A shutdown microprocessor resumes execution when NMI or RESET is asserted. Figure 3–15 illustrates a shutdown cycle using pipelined addressing.

Figure 3–15. TI486SXLC Pipelined Shutdown Cycle



3.4.6 Internal Cache Interface

The TI486SXLC cache is an 8K-byte write-through unified instruction/data cache with lines that are allocated only during memory read cycles. The cache is configured as two-way set associative. The cache organization consists of 1024 sets, each containing two lines of four bytes each.

3.4.6.1 Cache Fills

Any unlocked memory-read cycle can be cached by the TI486SXLC series microprocessor. The microprocessor does not automatically cache accesses to memory addresses specified by the Noncacheable-Region registers. Additionally, the KEN# input can be used to enable caching of memory accesses on a cycle-by-cycle basis. The microprocessor acknowledges the KEN# input only if the KEN enable bit is set in the CCR0 Configuration register.

As shown in Figure 3–16 and Figure 3–17, the microprocessor samples the KEN# input one CLK2 period before READY# is sampled active. If KEN# is asserted and the current address is cacheable, the microprocessor fills two bytes of a line in the cache with the data present on the data bus pins. The states of BHE# and BLE# are ignored if KEN# is asserted for the cycle.

Figure 3–16. TI486SXLC Nonpipelined Cache Fills Using KEN# (With Different Numbers of Wait States)

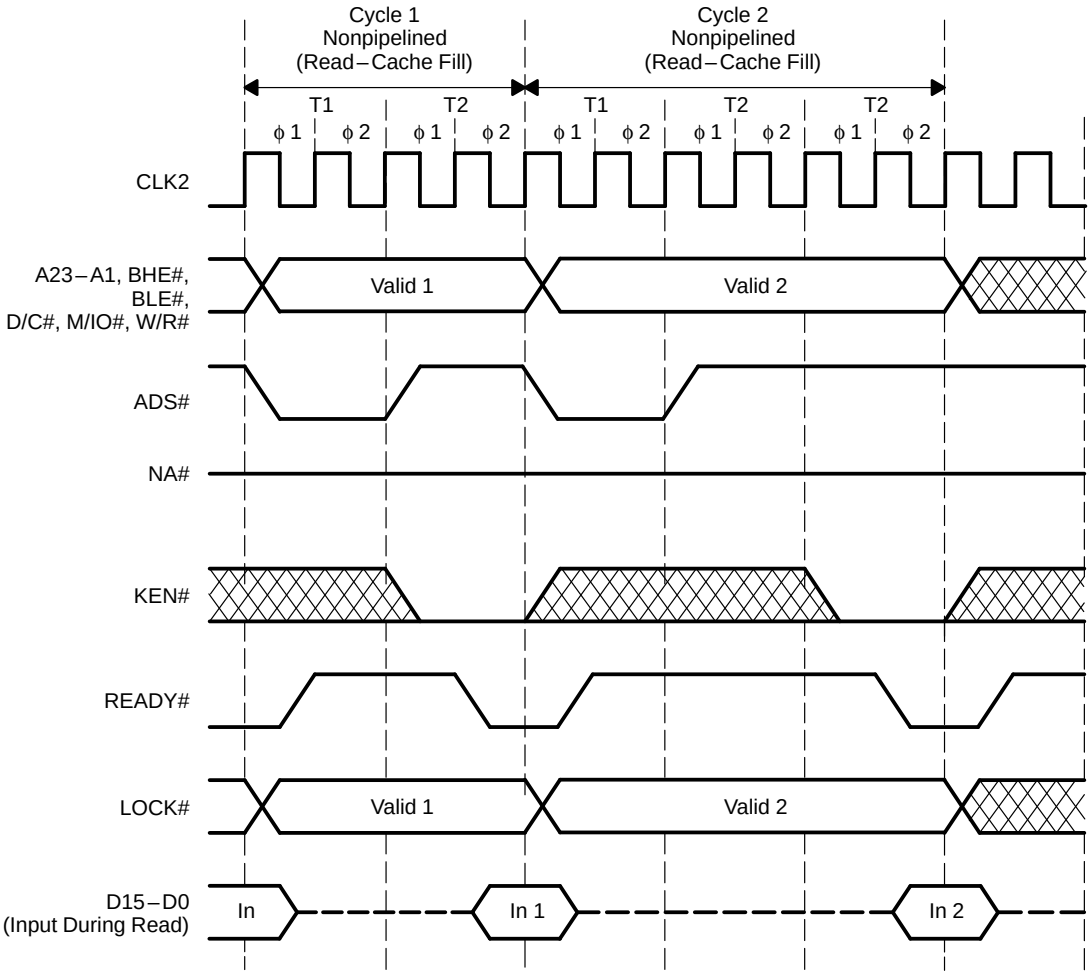
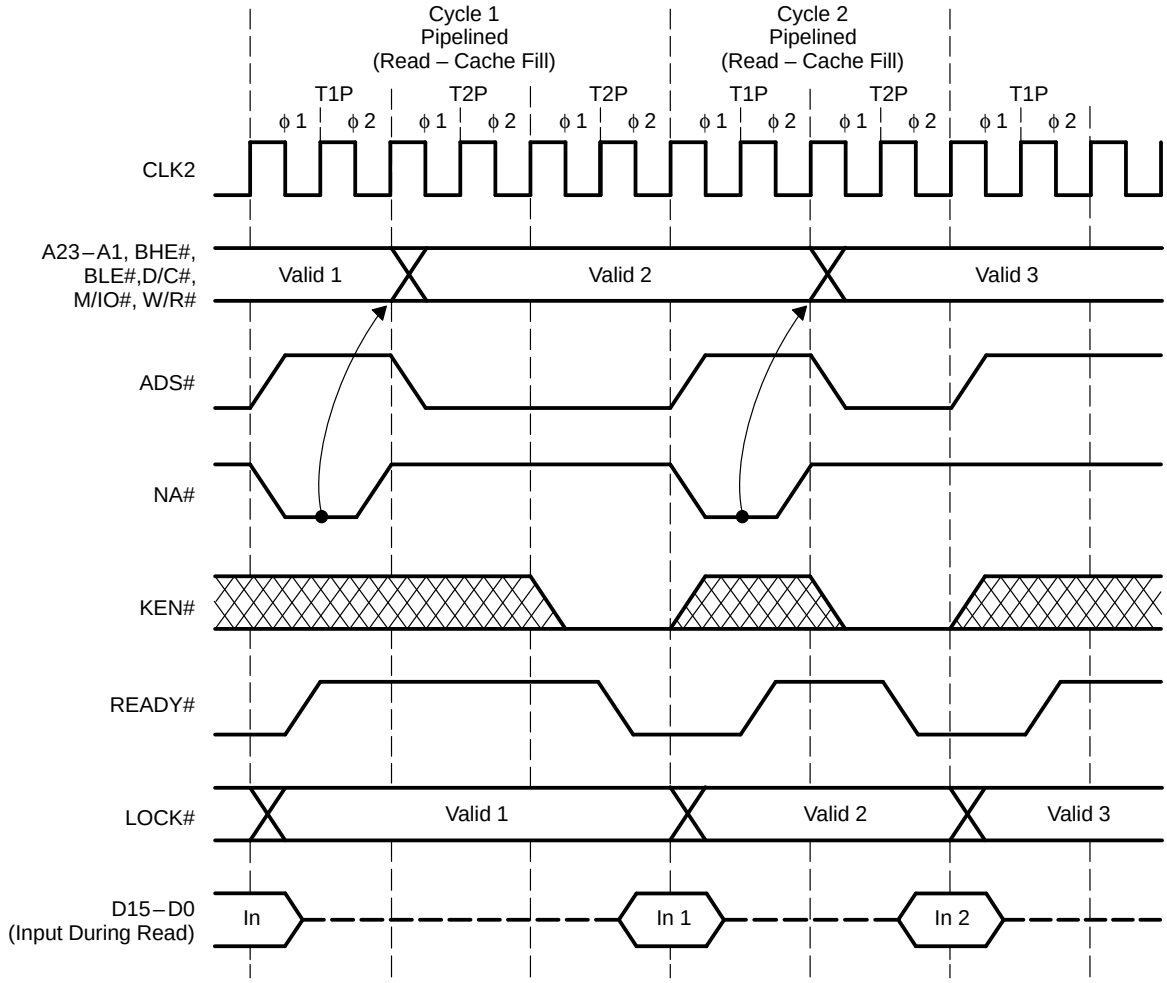


Figure 3–17. TI486SXLC Pipelined Cache Fills Using KEN# (With Different Numbers of Wait States)



3.4.6.2 Flushing the Cache

To maintain cache coherency with external memory, the TI486SXLC series microprocessor cache contents should be invalidated when previously cached data is modified in external memory by another bus master. The microprocessor invalidates the internal cache contents during execution of the INVD and WBINVD instructions following assertion of:

- ❑ HLDA if the BARB bit is set in the CCR0 Configuration register
- ❑ FLUSH# if the FLUSH bit is set in CCR0

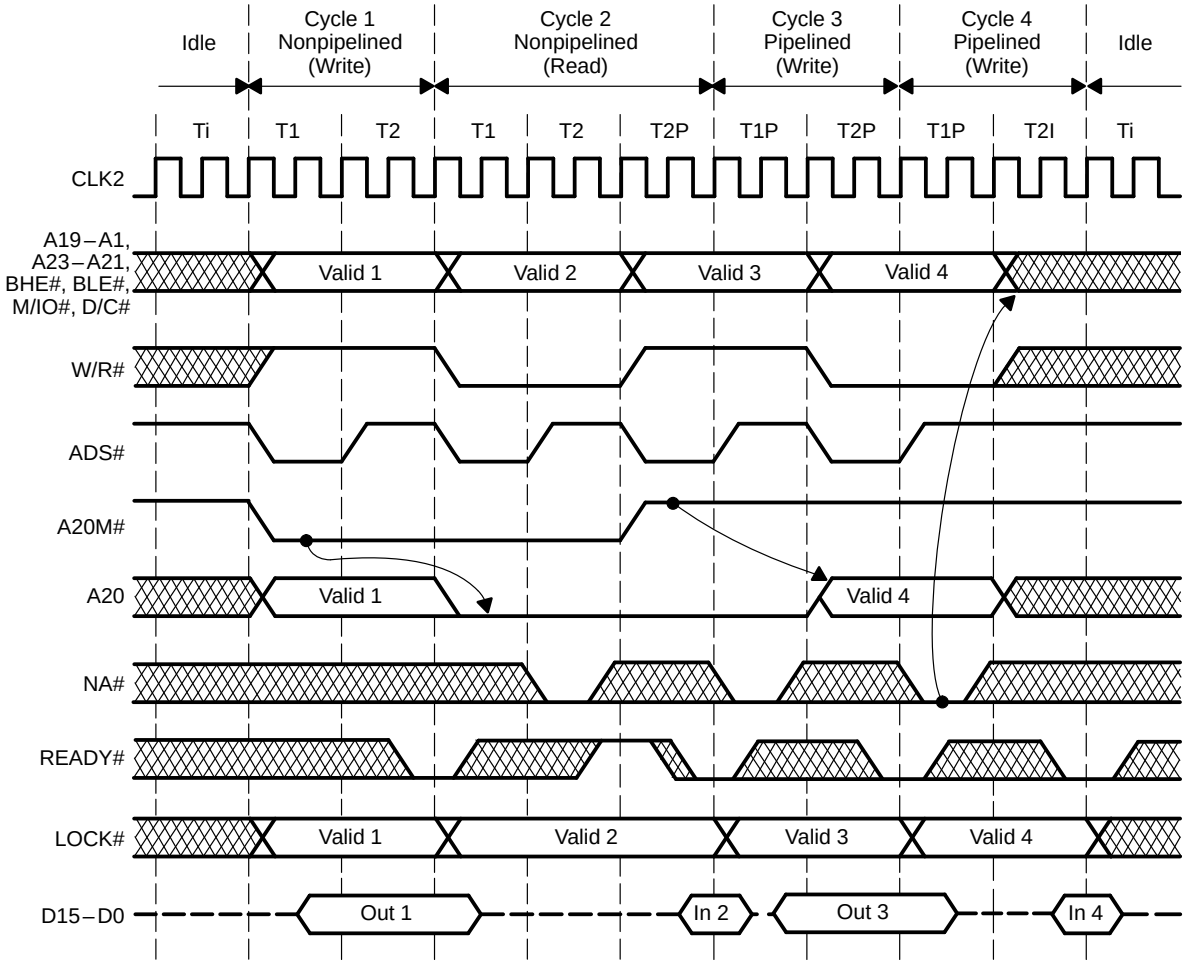
The microprocessor samples the FLUSH# input on the rising edge of CLK2 corresponding to the beginning of phase two ($\phi 2$) of the internal processor clock. If FLUSH# is asserted, the microprocessor invalidates the entire contents of the internal cache. The actual point in time when the cache is invalidated depends upon the internal state of the execution pipeline. FLUSH# must be asserted for at least two CLK2 periods and must meet specified setup and hold times to be recognized on a specific CLK2 edge.

3.4.7 Address Bit-20 Masking

The TI486SXLC series microprocessor can be forced to provide 8086 1M-byte address wraparound compatibility by setting the A20M bit in the CCR0 Configuration register and asserting the A20M# input. When the A20M# is asserted, the 20th bit in the address to both the internal cache and the external bus pin is masked (zeroed).

As shown in Figure 3–18, the microprocessor samples the A20M# input on the rising edge of CLK2 corresponding to the beginning of phase two ($\phi 2$) of the internal processor clock. If A20M# is asserted and paging is not enabled, the microprocessor masks the A20 signal internally starting with the next cache access and externally starting with the next bus cycle. If paging is enabled, the A20 signal is not masked regardless of the state of A20M#. The A20 signal remains masked until the access following detection of an inactive state on the A20M# pin. A20M# must be asserted for a minimum of two CLK2 periods and must meet specified setup and hold times to be recognized on a specific CLK2 edge.

Figure 3–18. TI486SXLC Masking A20 Using A20M# During Burst of Bus Cycles



An alternative to using the A20M# pin is to set the NC0 bit in the CCR0 Configuration register. When the NC0 bit is set, the microprocessor does not automatically cache accesses to the first 64K bytes and to 1M byte + 64K bytes. This prevents data within the wraparound memory area from residing in the internal cache and eliminates the need for masking address A20 to the internal cache.

3.4.8 Hold-Acknowledge State

The hold-acknowledge state lets an external device in a TI486SXLC microprocessor system acquire the system bus while the microprocessor is held in an inactive bus state. This allows external bus masters to take control of the microprocessor bus and directly access system hardware in a shared manner. The microprocessor continues to execute instructions out of the internal cache (if enabled) until a system bus cycle is required.

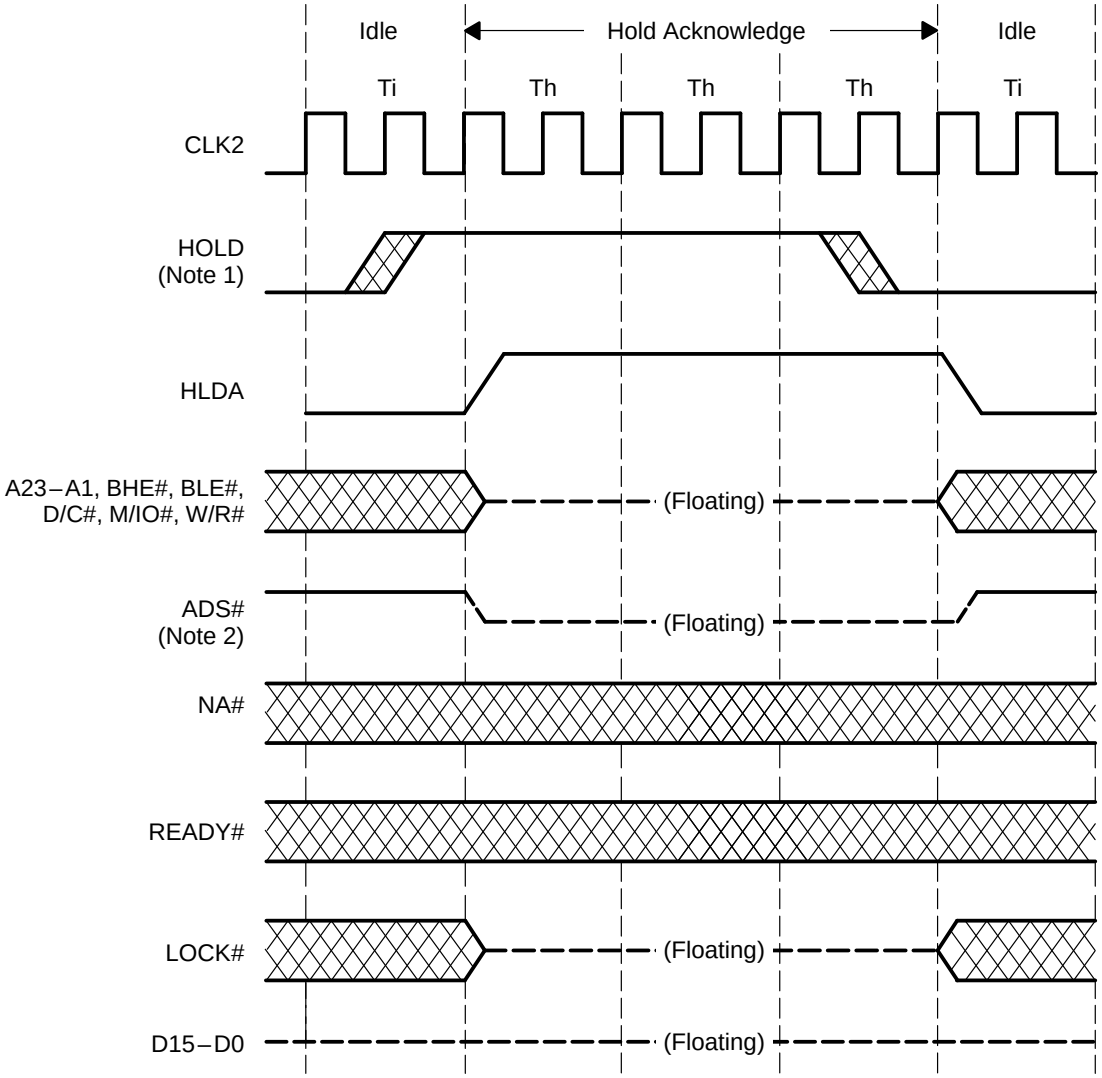
The hold-acknowledge state (Th) is entered in response to assertion of the HOLD input. In the hold-acknowledge state, the microprocessor floats all output and bidirectional signals, except for HLDA and SUSPA#. HLDA is asserted as long as the microprocessor remains in the hold-acknowledge state. All inputs except HOLD, FLUSH#, FLT#, SUSP# and RESET are ignored.

State Th can be entered directly from a bus-idle state, as in Figure 3–19, or after the completion of the current physical bus cycle if the LOCK signal is not asserted, as in Figure 3–20 and Figure 3–21. The CPU samples the HOLD input on the rising edge of CLK2 corresponding to the beginning of phase one ($\phi 1$) of the internal processor clock. HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold requirements are met in every bus state.

The hold-acknowledge state is exited in response to the HOLD input being negated. The next bus start is an idle state (Ti) if no bus request is pending, as in Figure 3–19. If an internal bus request is pending, as in Figure 3–20 and Figure 3–21, the next bus state is T1. State Th is also exited in response to RESET being asserted. If HOLD remains asserted when RESET goes inactive, the microprocessor enters the hold-acknowledge state before performing any bus cycles, provided HOLD is still asserted when the CPU is ready to perform its first bus cycle.

If a rising edge occurs on the edge-triggered NMI input while in state Th, the event is remembered as a nonmaskable interrupt 2 and is serviced when the state is exited.

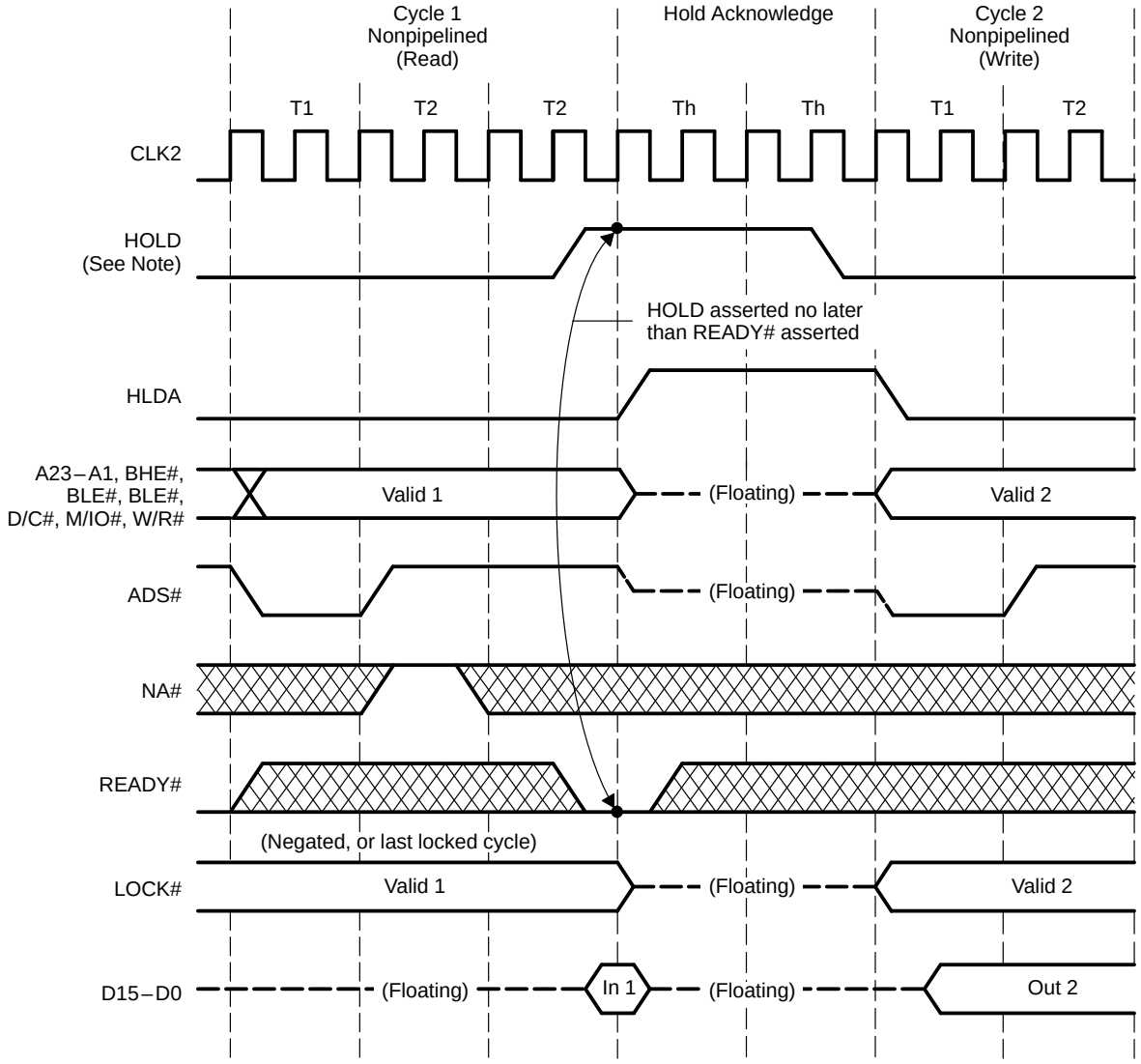
Figure 3–19. TI486SXLC Requesting Hold From Bus-Idle State



Notes: 1) HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold requirements are met in every bus state. Violating setup or hold requirements results in incorrect operation.

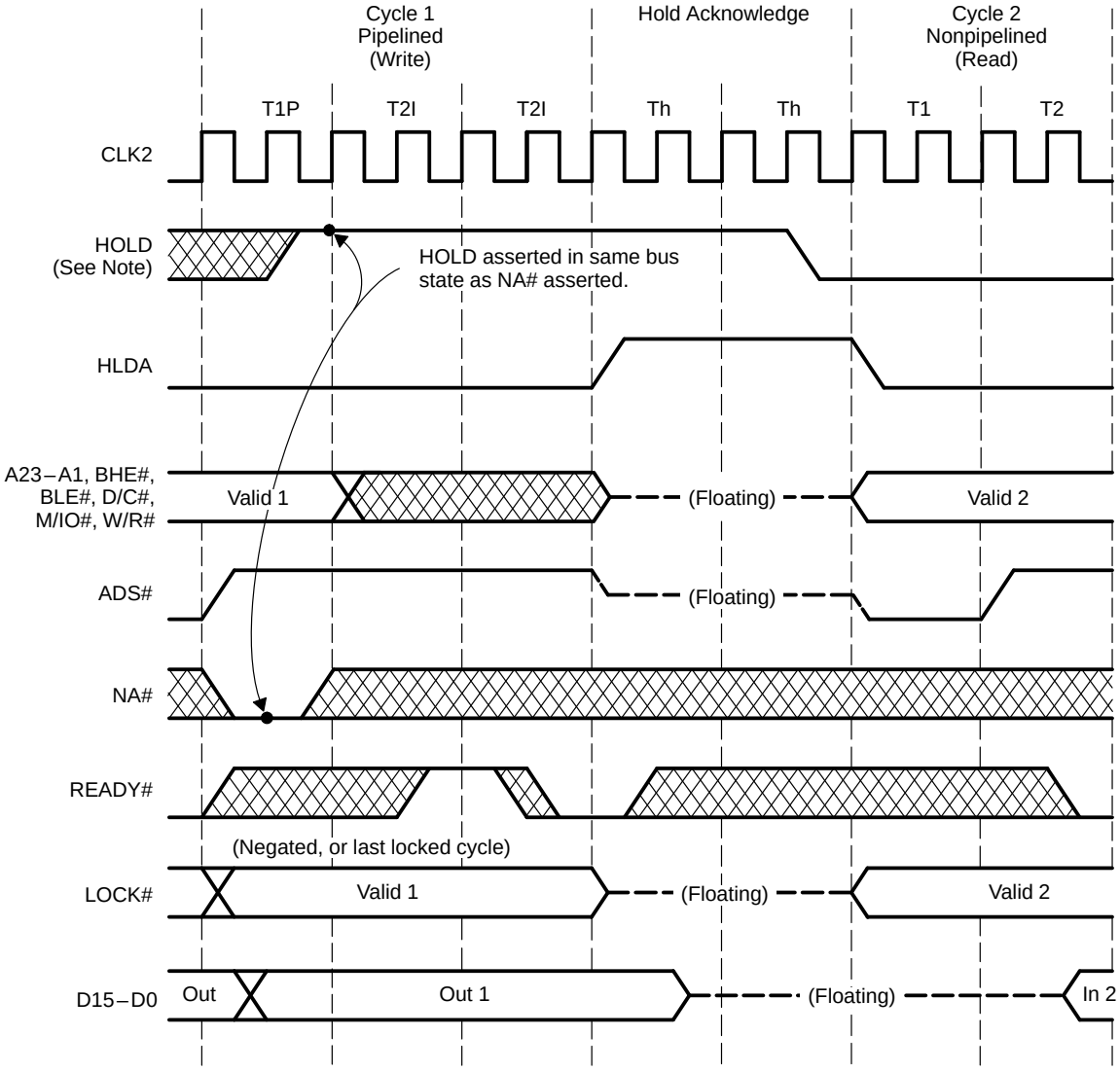
2) For maximum design flexibility, the CPU has no internal pullup resistors on its outputs. External pullups may be required on ADS# and other outputs to keep them negated during the hold-acknowledge period.

Figure 3–20. TI486SXLC Requesting Hold From Active Nonpipelined Bus



Note: HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold requirements are met in every bus state. Violating setup or hold requirements results in incorrect operation.

Figure 3–21. TI486SXLC Requesting Hold From Active Pipelined Bus



Note: HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold requirements are met in every bus state. Violating setup or hold requirements will result in incorrect operation.

3.4.9 Coprocessor Interface

The data-bus, address-bus, and bus-cycle-definition signals, and the coprocessor interface signals (PEREQ, BUSY#, and ERROR#), control communication between the TI486SXLC microprocessor and a coprocessor. The microprocessor decodes coprocessor or ESC opcodes and transfers the opcode and operands to the coprocessor via I/O port accesses. Address 80 00F8h functions as the control-port address, and 80 00FCh and 80 00FEh are used for operand transfers.

Coprocessor cycles can be read or write and can be nonpipelined or pipelined. Coprocessor cycles must be terminated by READY# and, as with any other bus cycle, can be terminated as early as the second bus state of the cycle.

BUSY#, ERROR#, and PEREQ are asynchronous level-sensitive inputs that synchronize CPU and coprocessor operation. All three signals are sampled at the beginning of phase one ($\phi 1$) and must meet specified setup and hold times to be recognized at a given CLK2 edge.

3.4.10 SMM Interface

System management mode (SMM) uses two TI486SXLC microprocessor pins, SMI# and SMADS#. The bidirectional SMI# pin is a nonmaskable interrupt that is a higher priority than the NMI input. SMI# must be active for at least four CLK2 periods to be recognized by the microprocessor. Once the microprocessor recognizes the active SMI# input, the CPU drives the SMI# pin low for the duration of the SMI service routine.

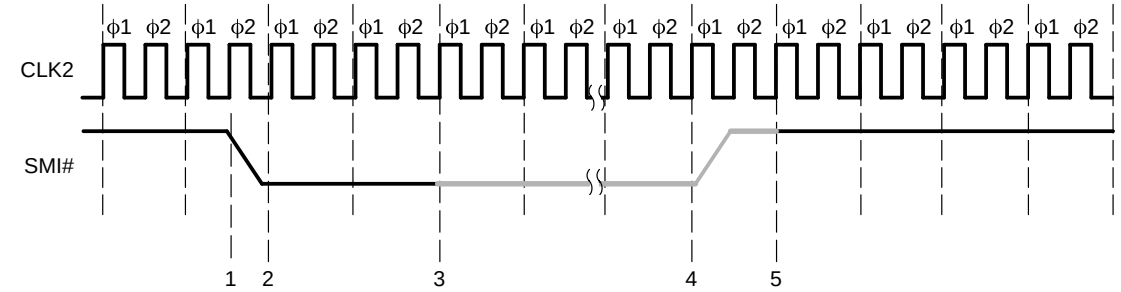
The SMADS# pin outputs the SMM address strobe that indicates an SMM memory bus cycle is in progress and a valid SMM address is on the address bus. The SMADS# functional timing, output delay times, and float delay times are identical to the main memory address strobe (ADS#) timing.

3.4.10.1 SMI Handshake

The functional timing for the SMI# interrupt is shown in Figure 3–22. Five significant events take place during an SMI# handshake:

1) The SMI# input pin is driven active (low) by the system logic.
2) The CPU samples SMI# active on the rising edge of CLK2 phase one ($\phi 1$).
3) Four CLK2 edges after sampling the SMI# active, the CPU switches the SMI# pin to an output and drives SMI# low.
4) Following execution of the RSM instruction, the CPU drives the SMI# pin high for two CLK2 edges indicating completion of the SMI service routine.
5) 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 the inactive (high) level after the pin has been changed to an input.

Figure 3–22. TI486SXLC SMI# Timing



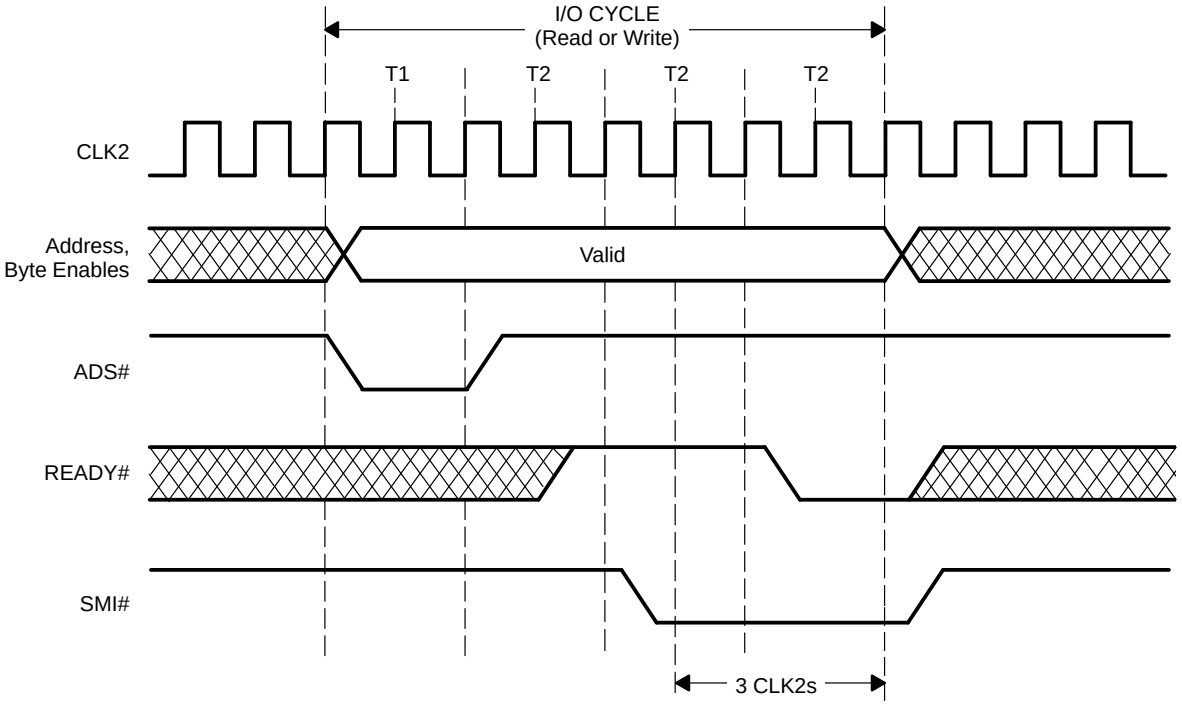
Indicates that TI486SXLC drives the SMI# pin.

3.4.10.2 I/O Trapping

The TI486SXLC series provides I/O trapping 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 three CLK2 edges before 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 3–23. The microprocessor immediately traps to the SMI interrupt handler following execution of the I/O instruction. No other instructions are executed between completing the I/O instruction and entering the SMI service routine. The I/O trap mechanism is not active during coprocessor accesses.

Figure 3–23. TI486SXLC I/O Trap Timing



3.4.11 Power Management

The power-management features in the TI486SXL(C) family of microprocessors allow a dramatic reduction in the current required when the microprocessor is in suspend mode (typically less than three percent of the operating current). Suspend mode is entered either by a hardware- or software-initiated action.

Using the hardware to initiate suspend mode involves a two-pin handshake using the SUSP# and SUSPA# signals. Using the software involves initiating the suspend mode through execution of the HALT instruction. Additional power management can be achieved by stopping and restarting the input clock. This technique is available because the TI486SXLC series microprocessors are static devices, meaning that the clock can be stopped and restarted without loss of any internal CPU data. See subsection 3.4.11.3, *Stopping the Input Clock*, on page 3-47.

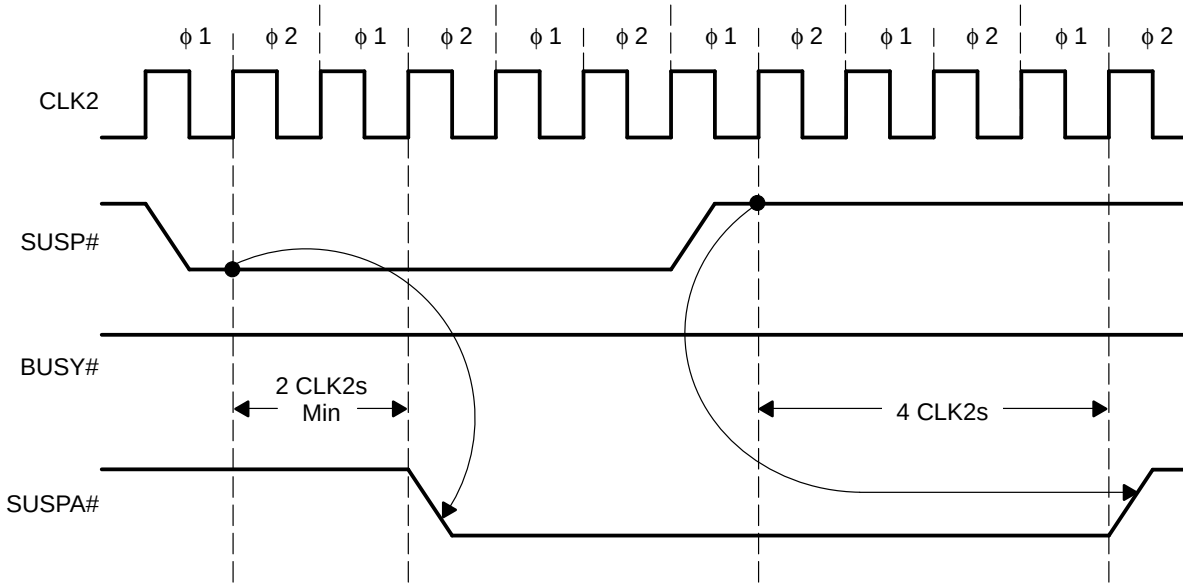
3.4.11.1 SUSP#-Initiated Suspend Mode

The TI486SXLC series microprocessor 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 microprocessor also waits for the coprocessor to indicate a not-busy status (BUSY#=1) before entering suspend mode. The SUSPA# output is then asserted. The microprocessor responds to SUSP# and asserts SUSPA# only if the SUSP bit is set in the CCR0 Configuration register.

Figure 3–24 illustrates the microprocessor functional timing for SUSP#-initiated suspend mode. SUSP# is sampled on the phase two ($\phi 2$) CLK2 rising edge and must meet specified setup and hold times to be recognized at a particular CLK2 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 two CLK2 periods. As a maximum, the microprocessor can execute up to two instructions and associated bus cycles before asserting SUSPA#. The time required for the microprocessor to deactivate SUSPA# once SUSP# has been sampled inactive is four CLK2 periods.

Figure 3–24. TI486SXLC SUSP#-Initiated Suspend Mode

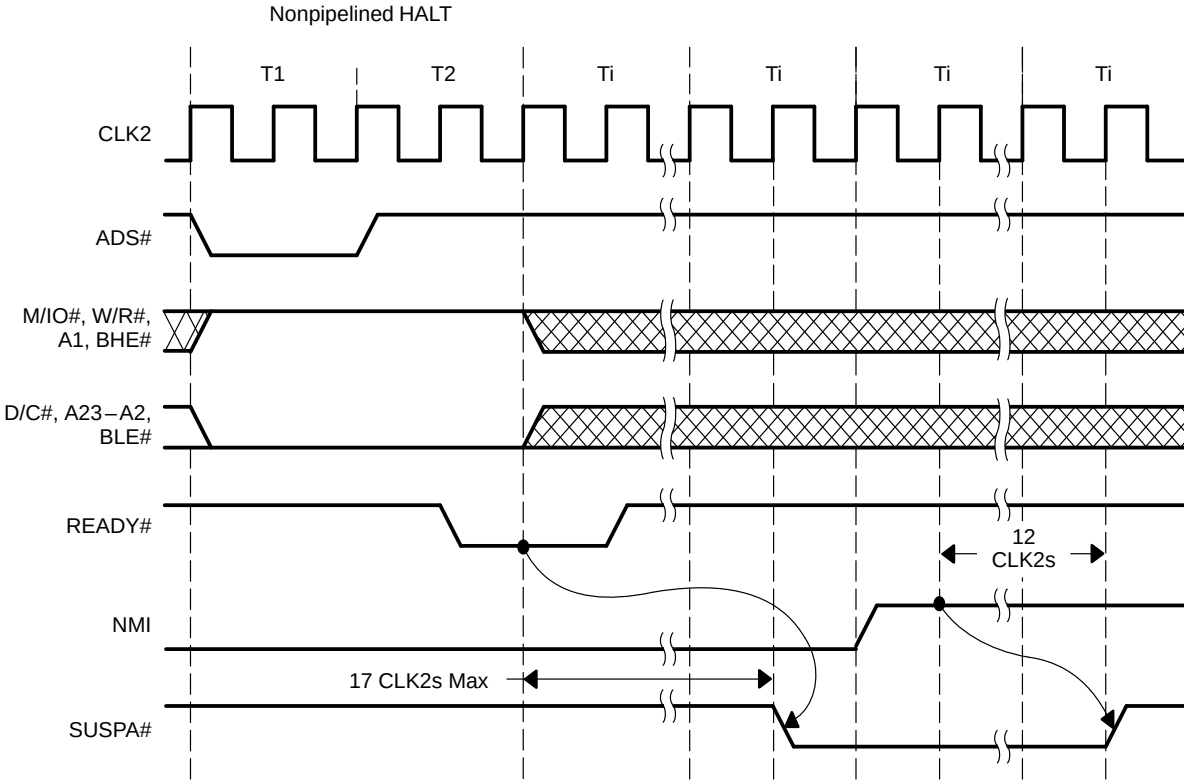


If the microprocessor is in a hold-acknowledge state and SUSP# is asserted, the processor may or may not enter suspend mode depending on the state of the microprocessor internal execution pipeline. If the microprocessor is in a SUSP#-initiated suspend state and the CLK2 input is not stopped, the processor recognizes and acknowledges the HOLD input. The microprocessor stores the occurrence of FLUSH#, NMI, and INTR (if enabled) for execution once suspend mode is exited.

3.4.11.2 HALT-Initiated Suspend Mode

The TI486SXLC series microprocessor also enters suspend mode as a result of executing a HALT instruction. The SUSPA# output is asserted no more than 17 CLK2 periods following a READY# sampled active for the HALT bus cycle as shown in Figure 3–25. Suspend mode is then exited upon recognition of an NMI or an unmasked INTR. SUSPA# is deactivated 12 CLK2 periods after sampling an active NMI or unmasked INTR. If the microprocessor is in a HALT-initiated suspend mode and the CLK2 input is not stopped, the processor recognizes and acknowledges the HOLD input. The microprocessor stores the occurrence of FLUSH# for execution once suspend mode is exited.

Figure 3–25. TI486SXLC HALT-Initiated Suspend Mode



3.4.11.3 Stopping the Input Clock

Because the TI486SXLC series microprocessors are static devices, the input clock (CLK2) can be stopped and restarted without loss of any internal CPU data. This assumes that the TI486SXLC2 microprocessor is in nonclock-doubled mode when the input clock is stopped. (Refer to subsection 3.2.1, *Clock Doubling Using Software Control*, page 3-13.) CLK2 can be stopped in either phase one ($\phi 1$) or phase two ($\phi 2$) of the clock and in either a logic-high or logic-low state. However, entering suspend mode before stopping CLK2 dramatically reduces the CPU current requirements. Therefore, the recommended sequence for stopping CLK2 in the TI486SXLC2 series microprocessor from clock-doubled mode is:

1) Bring the microprocessor out of clock-doubled mode
2) Initiate suspend mode
3) Wait for the microprocessor to assert SUSPA#
4) Stop the input clock

Note:

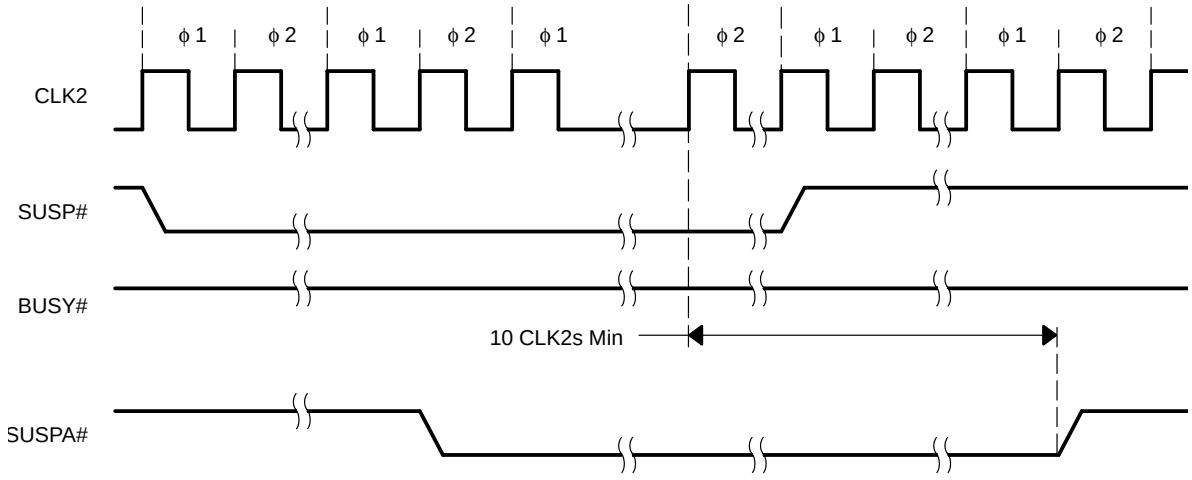
Suspend mode can be entered while in clock-doubled mode as long as CLK2 is not scaled or stopped.

For all other cases, including the TI486SXLC2 in nonclock-doubled mode, the recommended sequence is:

1) Initiate suspend mode
2) Wait for the microprocessor to assert SUSPA#
3) Stop the input clock

The TI486SXLC series microprocessor remains suspended until CLK2 is restarted and suspend mode is exited as described above. While CLK2 is stopped, the microprocessor can no longer sample and respond to any input stimulus including the HOLD, FLUSH#, NMI, INTR, and RESET inputs. Figure 3–26 illustrates the recommended sequence for stopping CLK2 using SUSP# to initiate suspend mode. CLK2 should be stable for a minimum of 10 clock periods before SUSP# is negated.

Figure 3–26. TI486SXLC Stopping CLK2 During Suspend Mode



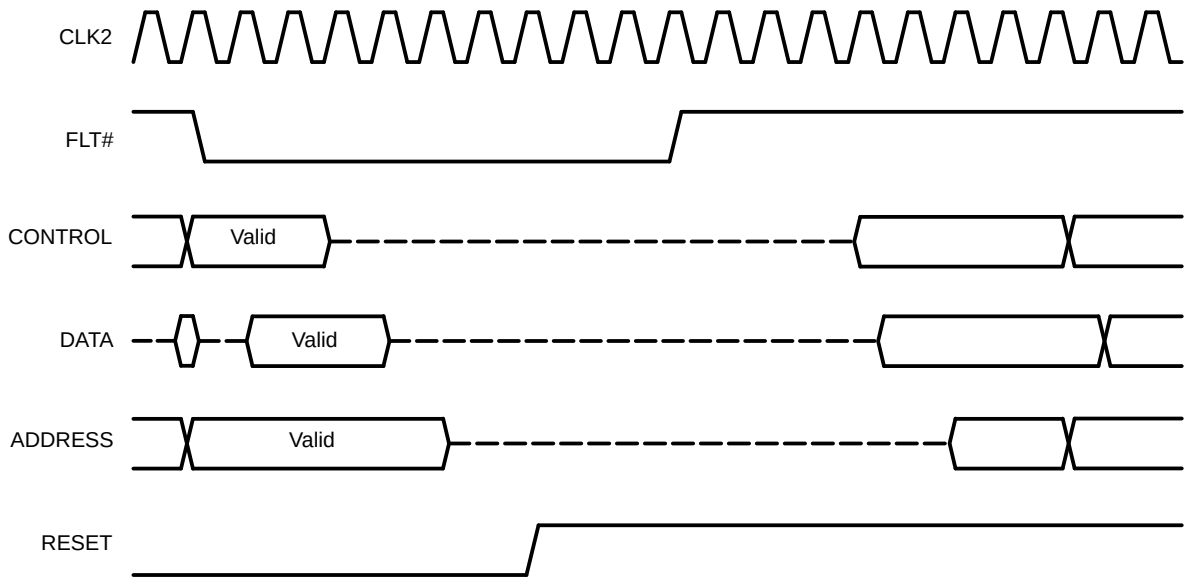
3.4.12 Float

Activating the FLT# input floats all TI486SXLC bidirectional and output signals. Asserting FLT# electrically isolates the microprocessor from the surrounding circuitry. This feature is useful in board-level test environments. Since the microprocessor is packaged in a surface-mount QFP, it is not usually socketed and cannot be removed from the motherboard when in-circuit emulation (ICE) is needed. Float capability allows connection of an emulator by clamping the emulator probe onto the microprocessor QFP without removing it from the circuit board.

FLT# is an asynchronous, active-low input. It is recognized on the rising edge of CLK2. When recognized, it aborts the current bus state and floats the outputs of the microprocessor as shown in Figure 3–27. FLT# must be asserted for a minimum of 16 CLK2 cycles. To exit the float condition, RESET should be asserted and held asserted until after FLT# is negated.

Asserting the FLT# input unconditionally aborts the current bus cycle and forces the microprocessor into the float mode. As a result, the microprocessor is not guaranteed to enter float in a valid state. After deactivating FLT#, the CPU is not guaranteed to exit float in a valid state. The microprocessor RESET input must be asserted before exiting float to ensure that the microprocessor is reset and that it returns in a valid state.

Figure 3–27. TI486SXLC Entering and Exiting Float



Chapter 4

TI486SXL Microprocessor Bus Interface

This chapter summarizes the TI486SXL series processor signals and describes all inputs/outputs, functional timing and bus operations (including pipelined and nonpipelined addressing), various interfaces, and power management.

Topic **Page**

4.1 Input/Output Signals

This section describes the TI486SXL series microprocessors' input and output signals. The discussion of these signals is arranged by the functional groups shown in Figure 4–1. Table 4–1 gives a brief description of each signal.

Figure 4–1. TI486SXL Functional Signal Groupings

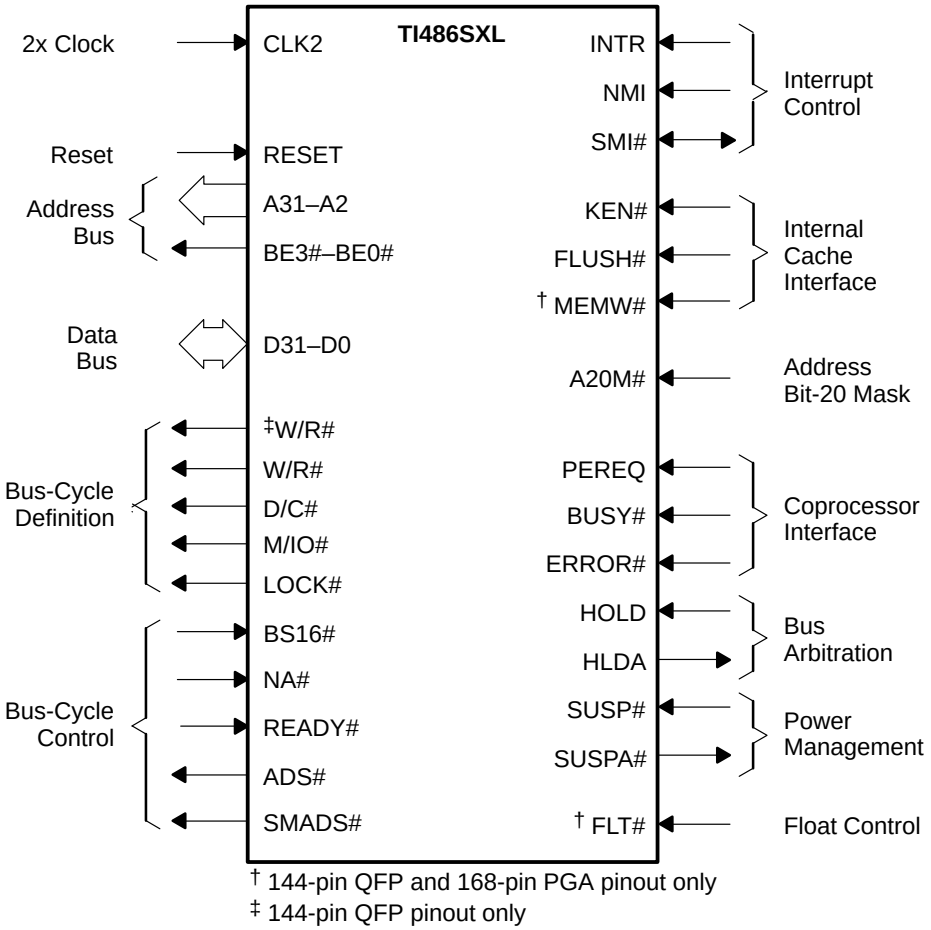


Table 4–1. TI486SXL Signal Summary

Signal	Signal Name	Signal Group
ADS#	Address strobe	Bus-cycle control
A20M#	Address bit-20 mask	None
A31–A2	Address bus lines	Address bus
BE3#–BE0#	Byte enables	Address bus
BS16#	Bus size 16	Bus-cycle control
BUSY#	Processor extension busy	Coprocessor interface
CLK2	2X clock input	None
D31–D0	Data bus	None
D/C#	Data/control	Bus-cycle definition
ERROR#	Processor extension error	Coprocessor interface
FLT#†	Float	None
FLUSH#	Cache flush	Internal cache interface
HLDA	Hold acknowledge	Bus arbitration
HOLD	Hold request	Bus arbitration
INTR	Maskable interrupt request	Interrupt control
KEN#	Cache enable	Internal Cache interface
LOCK#	Bus lock	Bus-cycle definition
MEMW#†	ISA memory write	Internal cache interface
M/IO#	Memory/input-output	Bus-cycle definition
NA#	Next address request	Bus-cycle control
NMI	Nonmaskable interrupt request	Interrupt control
PEREQ	Processor extension request	Coprocessor interface
READY#	Bus ready	Bus-cycle control
RESET	Reset	None
SMADS#	SMM address strobe	Bus-cycle control
SMI#	System management interrupt	Interrupt control
SUSP#	Suspend request	Power management
SUSPA#	Suspend acknowledge	Power management
W/R#‡	Write/read	Bus-cycle definition

† 144-pin QFP and 168-pin PGA pinout only.
‡ 144-pin QFP has W/R# on pins 36 and 37. These terminals must be connected together.

The following sections describe the signals and their functional characteristics. Additional signal information can be found in Chapter 5, *Electrical Specifications*. Chapter 5 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 TI486SXL series microprocessors.

4.1.1 TI486SXL Terminal Function Descriptions

Table 4–2 identifies and describes each of the TI486SXLC package terminals.

Table 4–2. TI486SXL Terminal Functions

Name	Terminal No. 132-pin	144-pin	168-pin	Description
A2	C4	73	Q14	Address Bus (active high). The address bus (A31–A2) signals are three-state outputs that provide addresses for physical memory and I/O ports. All address lines can be used to address physical memory, which allows a 4G-byte address space (0000 0000h to FFFF FFFFh). During I/O port accesses, A31–A16 are driven low (except for coprocessor accesses). This permits a 64-Kbyte I/O address space (0000 0000h to 0000 FFFFh). During all coprocessor I/O accesses, address lines A30–A16 are driven low and A31 is driven high. This allows A31 to be used by external logic to generate a coprocessor select signal. Coprocessor command transfers occur with address 8000 00F8h. Coprocessor data transfers occur with address 8000 00FCh. A31–A2 float while the CPU is in a hold-acknowledge or float state.
A3	A3	74	R15	
A4	B3	75	S16	
A5	B2	76	Q12	
A6	C3	77	S15	
A7	C2	78	Q13	
A8	C1	86	R13	
A9	D3	87	Q11	
A10	D2	88	S13	
A11	D1	89	R12	
A12	E3	90	S7	
A13	E2	93	Q10	
A14	E1	94	S5	
A15	F1	95	R7	
A16	G1	104	Q9	
A17	H1	106	Q3	
A18	H2	107	R5	
A19	H3	108	Q4	
A20	J1	109	Q8	
A21	K1	110	Q5	
A22	K2	113	Q7	
A23	L1	114	S3	
A24	L2	61	Q6	
A25	K3	60	R2	
A26	M1	59	S2	
A27	N1	58	S1	
A28	L3	84	R1	
A29	M2	83	P2	
A30	P1	82	P3	
A31	N2	81	Q1	
ADS#	E14	26	S17	Address Strobe (active low). This 3-state output indicates that the TI486SXL microprocessor has driven a valid address (A31–A2 and BE3#–BE0#) and bus-cycle definition (M/IO#, D/C#, and W/R#) on the appropriate output pins. During nonpipelined bus cycles, ADS# is active for the first clock of the bus cycle. During address pipelining, ADS# is asserted during the previous bus cycle and remains asserted until READY# is returned for that cycle. ADS# floats while the microprocessor is in a hold-acknowledge or float state.
A20M#	F13	43	D15	Address Bit-20 Mask (active low). This input causes the microprocessor to mask (force low) physical address bit 20 when driving the external address bus or performing an internal cache access. When the processor is in real mode, asserting A20M# emulates the 1M-byte address wraparound that occurs on the 8086. The A20 signal is never masked when paging is enabled regardless of the state of the A20M# input. The A20M# input is ignored following reset and can be enabled using the A20M bit in the CCR0 Configuration register. A20M# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.

Table 4–2. TI486SXL Terminal Functions (Continued)

Name	Terminal No. 132-pin	144-pin	168-pin	Description
BE3# BE2# BE1# BE0#	A13 B13 C13 E12	32 31 28 27	F17 J15 J16 K15	Byte Enables BE3#–BE0# (active low). These 3-state outputs determine which bytes within the 32-bit data bus are transferred during a memory or I/O access (Table 4–3). During a memory write, one or both of the upper bytes (D and C) of the data bus can be duplicated in the lower bytes (B and A) of the bus. This duplication is dependent on BE3#–BE0# as listed in Table 4–4. Generating A1–A0 using BE3#–BE0# is determined with the following equations: A0 = (BE0# • BE2#) + (BE0# • BE1#) A1 = BE0# • BE1# The relationship between A1–A0 and BE3#–BE0# is shown in Table 4–5.
BS16#	C14	115	C17	Bus Size 16 (active low). This input allows connection of the 32-bit microprocessor data bus to an external 16-bit data bus. When this input is activated, the microprocessor performs multiple bus cycles to couple read and write accesses from devices that cannot provide (accept) 32 bits of data in a single cycle. During bus cycles with BS16# active, data is transferred using data bus signals D15–D0 only.
BUSY#	B9	48	S4	Coprocessor Busy (active low). This input indicates to the TI486SXL that the coprocessor is currently executing an instruction and is unable to accept another opcode. When the microprocessor encounters a WAIT instruction or any coprocessor instruction that operates on the coprocessor stack (i.e., load, pop, or arithmetic operation), BUSY# is sampled. BUSY# is continually sampled and must be recognized as inactive before the CPU supplies the coprocessor with another instruction. However, coprocessor instructions FNINIT, FNCLEX are allowed to execute even if BUSY# is active because they are used for coprocessor initialization and exception clearing. BUSY# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.
CLK2	F12	25	C3	2X Clock Input (active high). This input signal is the basic timing reference for TI486SXL series microprocessors. The CLK2 input is internally divided by two to generate the internal processor clock. The external CLK2 is synchronized to a known phase of the internal processor clock by the falling edge of the RESET signal. External timing parameters are defined with respect to the rising edge of CLK2. For the TI486SXL2 microprocessors, the CLK2 input is used internally to generate the internal core processor clock and the internal bus interface clock. The external CLK2 is synchronized to a known phase of the internal processor clock by the falling edge of the RESET signal. External timing parameters are defined with respect to the rising edge of CLK2.

Table 4–2. TI486SXL Terminal Functions (Continued)

Name	Terminal No. 132-pin	Terminal No. 144-pin	Terminal No. 168-pin	Description
D/C#	A11	35	M15	Data/Control. This 3-state, bus-cycle-definition signal is low during control cycles and is high during data cycles. Control cycles are issued during functions such as a halt instruction, interrupt servicing, and code fetching. Data bus cycles include data access from either memory or I/O.
D0	H12	1	P1	Data Bus (active high). The data bus signals (D31–D0) are 3-state bidirectional signals that provide the data path between the microprocessor, the external memory, and the I/O devices. The data-bus inputs receive data during memory read, I/O read, and interrupt-acknowledge cycles and delivers output 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 float while the CPU is in a hold-acknowledge or float state.
D1	H13	144	N2	
D2	H14	143	N1	
D3	J14	137	H2	
D4	K14	136	M3	
D5	K13	135	J2	
D6	L14	134	L2	
D7	K12	133	L3	
D8	L13	131	F2	
D9	N14	130	D1	
D10	M12	129	E3	
D11	N13	128	C1	
D12	N12	127	G3	
D13	P13	118	D2	
D14	P12	117	K3	
D15	M11	116	F3	
D16	N11	124	J3	
D17	N10	123	D3	
D18	P11	122	C2	
D19	P10	121	B1	
D20	M9	102	A1	
D21	N9	101	B2	
D22	P9	100	A2	
D23	N8	99	A4	
D24	P7	3	A6	
D25	N6	4	B6	
D26	P5	142	C7	
D27	N5	141	C6	
D28	M6	12	C8	
D29	P4	13	A8	
D30	P3	14	C9	
D31	M5	15	B8	
ERROR#	A8	49	A12	Coprocessor Error (active low). This input indicates that the coprocessor generated an error during execution of an instruction. ERROR# is sampled by the microprocessor whenever a coprocessor instruction is executed. If ERROR# is sampled active, the processor generates exception 16, which is then serviced by the exception handling software. The following coprocessor instructions, which involve clear coprocessor error flags and save the coprocessor state, do not generate an exception 16 even if ERROR# is active: FNINIT, FNCLEX, FNSTSW, FNSTCW, FNSTENV, FNSAVE. ERROR# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.

Table 4–2. TI486SXL Terminal Functions (Continued)

Name	Terminal No. 132-pin	Terminal No. 144-pin	Terminal No. 168-pin	Description
FLT#	—	40	C11	Float (active low). This input forces all bidirectional and output signals to a 3-state condition. Floating the signals allows the microprocessor signals to be driven externally without physically removing the device from the circuit. The microprocessor must be reset following assertion or negation of FLT#. This signal may be used with an upgrade socket. FLT# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.
FLUSH#	E13	42	C15	Cache Flush (active low). This input invalidates (flushes) the entire cache. Use of FLUSH# to maintain cache coherency is optional. The cache may also be invalidated during each hold-acknowledge cycle by setting the BARB bit in the CCR0 Configuration register. The FLUSH# input is ignored following reset and can be enabled using the FLUSH bit in the CCR0 Configuration register. FLUSH# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.
HOLD	D14	7	E15	Hold Request (active high). This input indicates that another bus master requests control of the local bus. The bus arbitration (HOLD and HLDA) signals allow the microprocessor to relinquish control of its local bus when requested by another bus master device. Once the processor has relinquished its 3-stated bus, the bus master device can then drive the local bus signals. After recognizing the HOLD request and completing the current bus cycle or sequence of locked bus cycles, the microprocessor responds by floating the local bus and asserting the hold acknowledge (HLDA) output. Once HLDA is asserted, the bus remains granted to the requesting bus master until HOLD becomes inactive. When the microprocessor recognizes that HOLD is inactive, it simultaneously drives the local bus and drives HLDA inactive. External pullup resistors may be required on some of the microprocessor 3-state outputs to ensure that they remain inactive while in a hold-acknowledge state (or float state for the 144-pin QFP and 168-pin CPUs). The HOLD input is not recognized while RESET is active. If HOLD is asserted while RESET is active, RESET has priority, and the microprocessor places the bus into an idle state instead of a hold-acknowledge state. The HOLD input is also recognized during suspend mode provided that the CLK2 input has not been stopped. HOLD is level-sensitive and must meet specified setup and hold times for correct operation.

Table 4–2. TI486SXL Terminal Functions (Continued)

Name	Terminal No. 132-pin	Terminal No. 144-pin	Terminal No. 168-pin	Description
HLDA	M14	6	P15	Hold Acknowledge (active high). This output indicates that the microprocessor is in a hold-acknowledge state and has relinquished control of its local bus. While in the hold-acknowledge state, the microprocessor drives HLDA active and continues to drive SUSPA#, if enabled. The other microprocessor outputs are in the high-impedance state, allowing the requesting bus master to drive these signals. If the on-chip cache can satisfy bus requests, the microprocessor continues to operate during hold-acknowledge states. A20M# is internally recognized during this time. The microprocessor deactivates HLDA when the HOLD request is driven inactive. The microprocessor stores an NMI rising edge during a hold-acknowledge state for processing after HOLD is inactive. The FLUSH# input is also recognized during a hold-acknowledge state. If SUSP# is asserted during a hold-acknowledge state, the microprocessor may or may not enter suspend mode depending on the state of the internal execution pipeline. Table 4–6 summarizes the state of the microprocessor signals during hold acknowledge.
INTR	B7	53	A16	Maskable Interrupt Request. This level-sensitive input 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 Flag Word register IF bit. When unmasked, the microprocessor responds to the INTR input by issuing two locked interrupt-acknowledge cycles. To assure recognition of the INTR request, INTR must remain active until the start of the first interrupt-acknowledge cycle.
KEN#	B12	41	F15	Cache Enable (active low). This input indicates that the data being returned during the current cycle is cacheable. When KEN# is active and the microprocessor is performing a cacheable code fetch or memory data read cycle, the cycle is transformed into a cache fill. Use of the KEN# input to control cacheability is optional. The noncacheable region registers can also be used to control cacheability. Memory addresses specified by the noncacheable region registers cannot be cached regardless of the state of KEN#. I/O accesses, locked reads, SMM address space accesses, and interrupt-acknowledge cycles are never cached. During cached code fetches with BS16# asserted, two contiguous read cycles are performed to completely fill the 4-byte cache line. KEN# must be asserted during both read cycles to cause a cache line fill. If BS16# is inactive, only one bus cycle is required and KEN# must be asserted for the data to be cached. During memory data reads, the microprocessor performs as many read cycles as necessary to supply the required data to complete the current operation. Valid bits are maintained for each byte in the cache line and for each block of four lines, thus allowing data operands of less than four bytes to reside in the cache. If two read cycles are performed with the same address (A31–A2), KEN# must be asserted during both cycles to cache the data in these cycles. If the data is cached, the microprocessor ignores the state of the byte enables (BE3# – BE0#), and four bytes of data (2 bytes if BS16# is asserted) are cached. The KEN# input is ignored following reset and can be enabled using the KEN bit in the CCR0 Configuration register. KEN# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.

Table 4–2. TI486SXL Terminal Functions (Continued)

Name	Terminal No. 132-pin	144-pin	168-pin	Description
LOCK#	C10	38	N15	LOCK (active low). This 3-state, bus-cycle-definition signal is asserted to deny access to the CPU bus by other bus masters. The LOCK# signal may be explicitly activated during bus operations by including the LOCK prefix on certain instructions. LOCK# is always asserted during descriptor and page table updates, interrupt-acknowledge sequences, and when executing the XCHG instruction. The microprocessor does not enter the hold-acknowledge state in response to HOLD while the LOCK# output is active.
MEMW#	—	66	B16	Memory Write (active low). This input is used in the cache interface logic, which flushes the cache in systems that hold the CPU during DMA and MASTER cycles.
M/IO#	A12	34	N16	Memory/IO. This 3-state, bus-cycle-definition signal is low during I/O read and write cycles and is high during memory cycles.
NA#	D13	9	A13	Next Address Request (active low). This input requests address pipelining by the system hardware. When asserted, the system indicates that it is prepared to accept new bus-cycle definition and address signals (M/IO#, D/C#, W/R#, A31–A2, BS16#, and BE3#–BE0#) from the microprocessor even if the current bus cycle has not been terminated by assertion of READY#. If the microprocessor has an internal bus request pending and the NA# input is sampled active, the next bus-cycle definition and address signals are driven onto the bus.
NC†	B6	39 65 71 138	A3 A5 A14 A17 B14 B15 B17 C10 C12 C14 D16 D17 F1 G15 H3 H15 J17 L15 N3 Q15 Q16 Q17 R16	Make no external connection.

† Connecting or terminating (high or low) any NC terminal(s) may cause the microprocessor to produce unpredictable results or not operate.

Table 4–2. TI486SXL Terminal Functions (Continued)

Name	Terminal No. 132-pin	Terminal No. 144-pin	Terminal No. 168-pin	Description
NMI	B8	51	A15	Nonmaskable Interrupt Request. This rising-edge-sensitive input causes the processor to suspend execution of the current instruction stream and begin execution of an NMI interrupt service routine. The NMI interrupt service request cannot be masked by software. Asserting NMI causes an interrupt that internally supplies interrupt vector 2h to the CPU core. External interrupt-acknowledge cycles are not necessary since the NMI interrupt vector is supplied internally. Once NMI processing has started, no additional NMIs are processed until an IRET instruction is executed. The microprocessor samples NMI at the beginning of each phase two (ϕ2) clock period. To assure recognition, NMI must be inactive for at least eight CLK2 periods and then be active for at least eight CLK2 periods. Additionally, specified setup and hold times must be met to assure recognition at a particular clock edge.
PEREQ	C8	50	R17	Coprocessor Request (active high). This input indicates that the coprocessor is ready to transfer data to or from the CPU. The coprocessor can assert PEREQ in the process of executing a coprocessor instruction. The microprocessor internally stores the current coprocessor opcode and transfers the correct data to support coprocessor operations. The microprocessor employs PEREQ to synchronize the transfer of required operands. PEREQ is internally connected to a pulldown resistor to prevent this signal from floating active when left unconnected.
READY#	G13	10	F16	Ready (active low). This input is generated by the system hardware to indicate that the current bus cycle can be terminated. During a read cycle, assertion of READY# indicates that the system hardware has presented valid data to the CPU. When READY# is sampled active, the microprocessor latches the input data and terminates the cycle. During a write cycle, READY# assertion indicates that the system hardware has accepted the microprocessor output data. READY# must be asserted to terminate every bus cycle, including halt and shutdown indication cycles.
Reserved	—	—	A10	
RESET	C9	45	C16	Reset (active high). When asserted, RESET suspends all operations in progress and places the microprocessor into a reset state. RESET is a level-sensitive synchronous input and must meet specified setup and hold times to be properly recognized by the microprocessor. The microprocessor begins executing instructions at physical address location FF FFF0h approximately 400 CLK2 edges after RESET is driven inactive (low). While RESET is active, the microprocessor is initialized to nonclock-doubled mode (for the TI486SXL2) and all other input pins are ignored. The remaining signals are initialized to their reset state during the internal processor reset sequence. The reset signal states for the microprocessor are shown in Table 4–6.
SMADS#	C6	29	B13	SMM Address Strobe (active low). SMADS#, a 3-state output, 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.

Table 4–2. TI486SXL Terminal Functions (Continued)

Name	Terminal No. 132-pin	Terminal No. 144-pin	Terminal No. 168-pin	Description
SMI#	C7	67	B10	System Management Interrupt (active low). This 3-state, bidirectional, level-sensitive, input/output signal is an interrupt with higher priority than the NMI interrupt. SMI# must be active for at least four CLK2 clock periods to be recognized by the microprocessor. After the SMI is acknowledged, the SMI# pin is driven low by the microprocessor 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. SMI# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.
SUSP#	A4	63	C13	Suspend Request (active low). This input requests the microprocessor to enter suspend mode. After recognizing SUSP# as active, the processor completes execution of the current instruction, any pending decoded instructions, and associated bus cycles. In addition, the microprocessor waits for the coprocessor to indicate a not-busy status (BUSY# = 1) before entering suspend mode and asserting suspend acknowledgement (SUSPA#). SUSP# is internally connected to a pullup resistor to prevent it from floating active when left unconnected.
SUSPA#	B4	64	B12	Suspend Acknowledge (active low). This output indicates that the microprocessor has entered the suspend mode as a result of SUSP# assertion or execution of a HALT instruction.
V_{CC}	A1 A5 A7 A10 A14 C5 C12 D12 G2 G3 G12 G14 L12 M3 M7 M13 N4 N7 P2 P8	5 11 16 17 30 44 52 55 56 62 68 79 85 91 98 103 105 119 125 132 139	B7 B9 B11 C4 C5 E2 E16 G2 G16 H16 K2 K16 L16 M2 M16 P16 R3 R6 R8 R9 R10 R11 R14	Power Supply. All pins must be connected and used.
VCC5	—	47	J1	5-V Power Supply

Table 4–2. TI486SXL Terminal Functions (Continued)

Name	Terminal No. 132-pin	144-pin	168-pin	Description
V_{SS}	A2 A6 A9 B1 B5 B11 B14 C11 F2 F3 F14 J2 J3 J12 J13 M4 M8 M10 N3 P6 P14	2 8 18 19 20 21 22 23 24 33 48 54 57 69 70 72 80 92 96 97 111	A7 A9 A11 B3 B4 B5 E1 E17 G1 G17 H1 H17 K1 K17 L1 L17 M1 M17 P17 Q2 R4 S6 S8 S9 S10 S11 S12 S14	Ground Pins. All pins must be connected and used.
W/R#	B10	36 37	N17	Write/Read. This 3-state, bus-cycle-definition signal is low during read cycles (data is read from memory or I/O) and is high during write bus cycles (data is written to memory or I/O).

4.1.2 Byte Enable Line Definitions

These 3-state outputs determine which bytes within the 32-bit data bus are transferred during a memory or I/O access. See Table 4–3.

Table 4–3. Byte Enable Line Definitions

Byte Enable Line	Byte Transferred
BE0#	D7–D0
BE1#	D15–D8
BE2#	D23–D16
BE3#	D31–D24

4.1.3 Write Duplication as a Function of BE3# – BE0#

During a memory write, one or both of the upper bytes (D and C) of the data bus can be duplicated in the lower bytes (B and A) of the bus. This duplication is dependent on BE3#–BE0# as listed in Table 4–4.

Table 4–4. Write Duplication as a Function of BE3#–BE0#

BE3#–BE0#	D31–D24	D23–D16	D15–D8	D7–D0	Duplicated Data
0000	D	C	B	A	No
0001	D	C	B	X	No
0011	D	C	D	C	Yes
0111	D	X	D	X	Yes
1000	X	C	B	A	No
1001	X	C	B	X	No
1011	X	C	X	C	Yes
1100	X	X	B	A	No
1101	X	X	B	X	No
1110	X	X	X	A	No

Note: BE3# – BE0# combinations not listed do not occur during TI486SXL bus cycles.
A = logical write data D7 – D0
B = logical write data D15 – D8
C = logical write data D23 – D16
D = logical write data D31 – D24
X = Don't care

4.1.4 Generating A1 – A0 Using BE3# – BE0#

Generating A1–A0 using BE3#–BE0# is determined with the following equations:

$$A0 = (BE0\# \bullet BE2\#) + (BE0\# \bullet BE1\#)$$

$$A1 = BE0\# \bullet BE1\#$$

The relationship between A1–A0 and BE3#–BE0# is shown in Table 4–5.

Table 4–5. Generating A1–A0 Using BE3#–BE0#

A31–A2	A1	A0	BE3#	BE2#	BE1#	BE0#
———	0	0	X	X	X	0
———	0	1	X	X	0	1
———	1	0	X	0	1	1
———	1	1	0	1	1	1

Note: X = Don't care

4.1.5 Signal States During Reset and Hold Acknowledge

RESET is the highest priority input signal. When RESET is asserted, the microprocessor aborts any current bus cycle and establishes real-mode

bus-cycle definition with active buses. See Table 3–3 and Section 4.3, *Reset Timing and Internal Clock Synchronization*, page 4-19.

The microprocessor enters the hold-acknowledge state in response to assertion of the HOLD input. During the hold-acknowledge state, the microprocessor floats all output and bidirectional signals, except for HLDA and SUSPA#. In the hold-acknowledge state, all inputs except HOLD, FLUSH#, FLT#, SUSP# and RESET are ignored. See Table 3–3 and subsection 4.4.9, *Hold Acknowledge State*, page 4-45. The hold-acknowledge state lets an external device acquire the system bus.

Table 4–6. TI486SXL Signal States During RESET and Hold Acknowledge

Signal Name	Signal State During Reset	Signal State During Hold Acknowledge
A20M#	Ignored	Input recognized
A31–A2	1	Float
ADS#	1	Float
BE3#–BE0#	0	Float
BS16#	Ignored	Ignored
BUSY#	Initiates self test	Ignored
D31–D0	Float	Float
D/C#	1	Float
ERROR#	Ignored	Ignored
FLT# †	Input recognized	Input recognized
FLUSH#	Ignored	Input recognized
HLDA	0	1
HOLD	Ignored	Input recognized
INTR	Ignored	Input recognized
KEN#	Ignored	Ignored
LOCK#	1	Float
MEMW# †	Ignored	Input recognized
M/IO#	0	Float
NA#	Ignored	Ignored
NMI	Ignored	Input recognized
PEREQ	Ignored	Ignored
READY#	Ignored	Ignored
RESET	Input recognized	Input recognized
SMADS#	Float	Float
SMI#	Ignored	Input recognized
SUSP#	Ignored	Input recognized
SUSPA#	Float	Driven
W/R#‡	0	Float

† 144-pin QFP and 168-pin PGA only
‡ 144-pin QFP has W/R# on pins 36 and 37. These terminals must be connected together.

4.2 Bus-Cycle Definition

The bus-cycle-definition signals consist of four 3-state outputs (M/IO#, D/C#, W/R#, and LOCK#) that define the type of bus-cycle operation. Table 4–7 defines the bus cycle for the possible states of these signals. M/IO#, D/C#, and W/R# are the primary bus-cycle-definition signals and are driven valid as ADS# (Address Strobe) becomes active. During nonpipelined cycles, the LOCK# output is driven valid along with M/IO#, D/C#, and W/R#. During pipelined addressing, LOCK# is driven at the beginning of the bus cycle, which is after ADS# becomes active for that cycle. The bus-cycle-definition signals are active low and float while the microprocessor is in a hold-acknowledge or float state.

Table 4–7. TI486SXL 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	X	—
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	Halt: A31–A2 = 0h and BE3#–BE0# = 1011 Shutdown: A31–A2 = 0h and BE3#–BE0# = 1110
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

4.2.1 Clock Doubling Using Software Control

The clock-doubled feature of the TI486SXL2 is enabled/disabled using Configuration Control register 0 (CCR0) bit 6. The following software code sets and resets CKD:

Set CKD programming sequence:

```
mov   al, 0C0h          ;select CCR0
out   22h, al
in    al, 23h           ;read CCR0
mov   ah, al            ;save in AH
or    ah, 40h           ;set AH<6>
mov   al, 0C0h          ;select CCR0
out   22h, al
mov   al, ah
out   23h, al           ;write CCR0
```

Reset CKD programming sequence:

```
mov   al, 0C0h          ;select CCR0
out   22h, al
in    al, 23h           ;read CCR0
mov   ah, al            ;save in AH
and   ah, 0BFh          ;reset AH<6>
mov   al, 0C0h          ;select CCR0
out   22h, al
mov   al, ah
out   23h, al           ;write CCR0
```

4.2.1.1 Entering Clock-Doubled Mode

The TI486SXL2 microprocessors power up in the nonclock-doubled mode. To enter the clock-doubled mode, set CLK2 to the desired frequency inside the phase-locked loop (PLL) lock range (see Table 5–5 and Table 5–6) and issue the set-CKD-programming sequence. Approximately 20 μs after the final OUT instruction has exited the processor pipeline, the PLL locks and the CPU enters clock-doubled mode. Until the PLL is locked, the processor continues to operate in the nonclock-doubled mode.

4.2.1.2 Clock-Scaling Sequence

To scale or stop CLK2 input when the processor is in clock-doubled mode, issue the reset-CKD-programming sequence. The final OUT instruction exiting the processor pipeline resets the CKD bit and puts the microprocessor into nonclock-doubled mode. This must occur before scaling or stopping the CLK2 input to prevent a synchronization error. This may be ensured by issuing a JUMP instruction, such as JMP $+2, before scaling CLK2.

To return the processor to clock-doubled mode, set CLK2 to the desired frequency inside the PLL lock range and issue the set-CKD-programming sequence. Approximately 20 μs after the final OUT instruction has exited the processor pipeline, the PLL locks and the processor enters clock-doubled mode.

4.2.1.3 Suspend Mode

Suspend mode can be initiated when the TI486SXL2 microprocessor is in clock-doubled mode as long as the CLK2 input is not scaled or stopped. Suspend mode does not disable the PLL; instead, changing the CLK2 frequency causes the PLL to lose lock.

For more detailed information on entering and exiting suspend in nonclock-doubled mode, refer to subsection 4.2.2, *Power Management*.

To get the lowest possible power state, bring the microprocessor out of clock-doubled mode, enter the suspend mode (using software or hardware), and stop the CLK2 input.

4.2.2 Power Management

The power-management signals allow the TI486SXL series microprocessors to enter suspend mode. Suspend-mode circuitry allows the microprocessor to consume minimal power while maintaining the entire internal CPU state.

4.2.2.1 Suspend Request (SUSP#)

Suspend Request (SUSP#) is an active-low input that requests the TI486SXL series microprocessors to enter suspend mode. For TI486SXL2 microprocessors, follow the clock-scaling sequence procedure in subsection 4.2.1 to enter nonclock-doubled mode before scaling or stopping the CLK2 input.

After recognizing SUSP# is active, the processor completes execution of the current instruction, any pending decoded instructions, and associated bus cycles. In addition, the microprocessor waits for the coprocessor to indicate a not-busy condition (BUSY#=1) before entering suspend mode and asserting suspend acknowledge (SUSPA#). During suspend mode, internal clocks are stopped and only the logic for monitoring RESET, HOLD, and FLUSH# remains active. With SUSPA# asserted, the CLK2 input to the microprocessor can be stopped in either phase. Stopping the CLK2 input further reduces current required by the microprocessor.

To resume operation, restart the CLK2 input (if stopped) and negate the SUSP# input. The TI486SXL2 processors can enter clock-doubled mode (subsection 4.2.1.1, *Entering Clock-Doubled Mode*) once the CLK2 input reaches the desired frequency within the PLL lock range. The processor then resumes instruction fetching and begins execution in the instruction stream at the point where it 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 using the SUSP bit in the CCR0 Configuration register.

4.2.2.2 Suspend Acknowledge (SUSPA#)

The Suspend Acknowledge (SUSPA#) output indicates that the TI486SXL series microprocessor has entered the suspend mode as a result of SUSP# assertion or execution of a HALT instruction. If SUSPA# is asserted and the CLK2 input is switching, the microprocessor continues to recognize RESET, HOLD, and FLUSH#. In addition, the TI486SXL2 microprocessor may stay in clock-doubled mode while the CLK2 input is switching. If suspend mode was entered as the result of a HALT instruction, the microprocessor also continues to monitor the NMI input, the SMI# input, and the unmasked INTR input. Detection of SMI#, INTR, or NMI forces the microprocessor to exit suspend mode and begin execution of the appropriate interrupt service routine. The CLK2 input to the processor can be stopped after SUSPA# has been asserted to reduce the power requirement of the microprocessor further. For this case, the TI486SXL2 microprocessor must be brought out of clock-doubled mode before stopping the CLK2 input to prevent a synchronization error. The SUSPA# output is disabled (floated) following reset and can be enabled using the SUSP bit in the CCR0 Configuration register.

Table 4–8 shows the state of the TI486SXL series microprocessor signals when the device is in suspend mode.

Table 4–8. TI486SXL Signal States During Suspend Mode

Signal Name	Signal State During Hold Acknowledge	Signal State During Halt-Initiated Suspend Mode
A20M#	Ignored	Ignored
A31–A2	1	1
ADS#	1	1
BE3#–BE0#	0	0
BS16#	Ignored	Ignored
BUSY#	Ignored	Ignored
D31–D0	Float	Float
D/C#	1	1
ERROR#	Ignored	Ignored
FLT# †	Input recognized	Input recognized
FLUSH#	Input recognized	Input recognized
HLDA	0	0
HOLD	Input recognized	Input recognized
INTR	Latched	Input recognized
KEN#	Ignored	Ignored
LOCK#	1	1
MEMW# †	Input recognized	Input recognized
M/IO#	0	0
NA#	Ignored	Ignored
NMI	Latched	Input recognized
PEREQ	Ignored	Ignored
READY#	Ignored	Ignored
RESET	Input recognized	Input recognized
SMADS#	1	1
SMI#	Latched	Input recognized
SUSP#	Input recognized	Ignored
SUSPA#	0	0
W/R#‡	0	0

† 144-pin QFP and 168-pin PGA only
‡ 144-pin QFP has duplicate W/R# inputs on pins 36 and 37

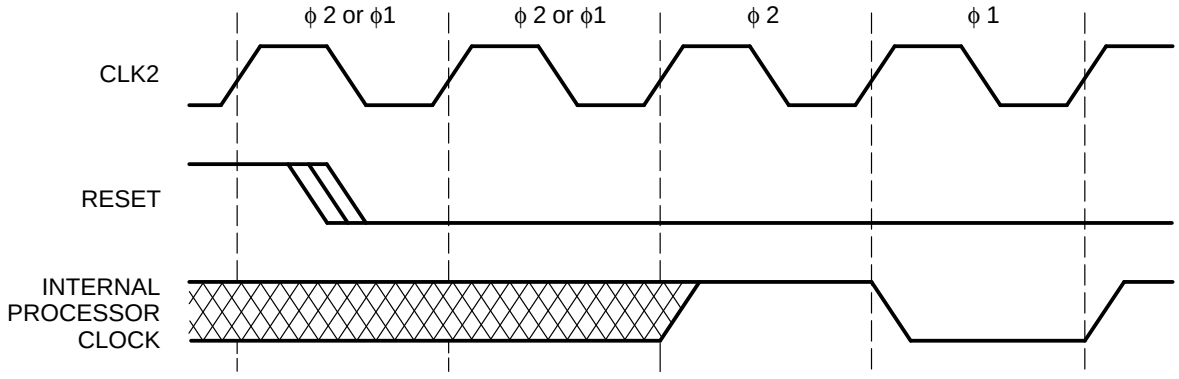
4.3 Reset Timing and Internal Clock Synchronization

RESET is the highest priority input signal and interrupts any processor activity when it is asserted. When RESET is asserted, the microprocessor aborts any bus cycle. Idle, hold-acknowledge, and suspend states are also discontinued and the reset state is established. RESET is used when the microprocessor is powered up to initialize the CPU to a known valid state and to synchronize the internal CPU clock with external clocks. The TI486SXL2 microprocessors are initialized to nonclock-doubled mode when RESET goes active.

RESET must be asserted for at least 15 CLK2 periods to ensure recognition by the microprocessor. If the self-test feature is to be invoked, RESET must be asserted for at least 80 CLK2 periods. RESET pulses of less than 15 CLK2 periods may not have sufficient time to propagate throughout the microprocessor and may not be recognized. RESET pulses of less than 80 CLK2 periods followed by a self-test request may incorrectly report a self-test failure when none has occurred.

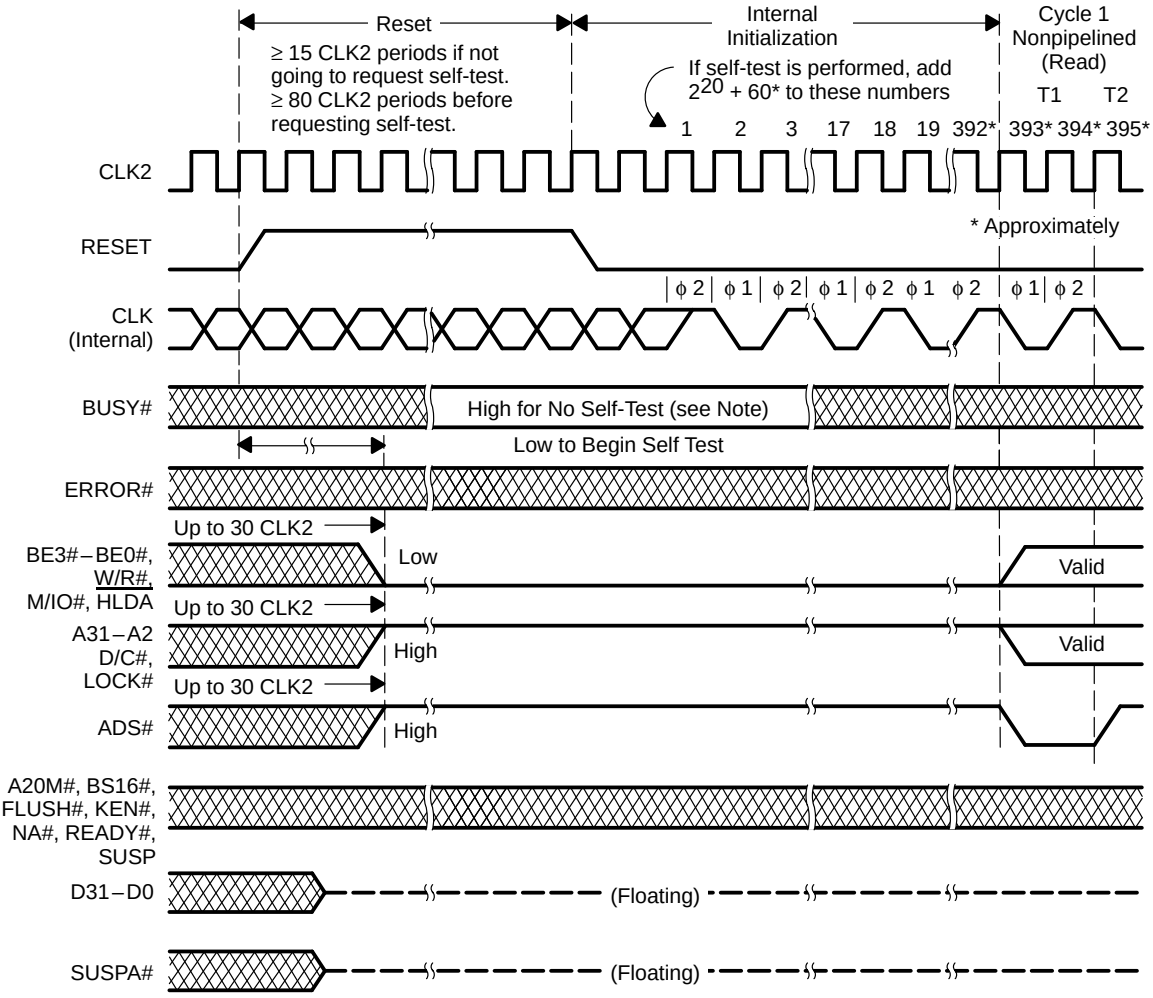
If the RESET falling edge meets specified setup and hold times, the internal processor clock phase is synchronized as illustrated in Figure 4–2. The TI486SXL internal processor clock is half the frequency of the CLK2 input and each CLK2 cycle corresponds to an internal CPU clock phase (ϕ). Phase two ($\phi 2$) of the internal clock is defined as the second rising edge of CLK2 following the falling edge of RESET. The TI486SXL2 internal core clock is the same frequency as the CLK2 input, and the internal bus interface clock is half the frequency of the CLK2 input. Phase two of the internal clock is defined as the second rising edge of CLK2 following the falling edge of RESET.

Figure 4–2. TI486SXL Internal Processor Clock Synchronization



Following the falling edge of RESET (and after self-test if it was requested), the microprocessor performs an internal initialization sequence for approximately 400 CLK2 periods. The microprocessor self-test feature is invoked if the BUSY# input is in the active (low) state when RESET falls inactive. The self-test sequence requires approximately (2^{20} + 60) CLK2 periods to complete. Even if the self-test indicates a problem, the microprocessor attempts to proceed with the reset sequence. Figure 4–3 illustrates the bus activity and timing during the microprocessor reset sequence.

Figure 4–3. TI486SXL Bus Activity From RESET Until First Code Fetch



Note: BUSY# should be held stable for 80 CLK2 periods before and after the CLK2 period in which RESET falling edge occurs.

Upon completion of self-test, the EAX register contains 0000 0000h if the microprocessor passed its internal self-test with no problems. Any nonzero value in the EAX register indicates that the microprocessor is faulty.

4.4 Bus Operation and Functional Timing

The TI486SXL series microprocessor communicates with the external system through separate, parallel buses for data and address. This is commonly called a demultiplexed address/data bus. This demultiplexed bus eliminates the need for address latches required in multiplexed address/data bus configurations, where the address and data are presented on the same pins at different times.

TI486SXL series microprocessor instructions can act on memory data operands consisting of 8-bit bytes, 16-bit words, or 32-bit double words. The microprocessor bus architecture allows for bus transfers of these operands without restrictions on physical address alignment. Any byte boundary may require more than one bus cycle to transfer the operand. This feature is transparent to the programmer.

The microprocessor data bus (D31–D0) is a bidirectional bus that can be configured as either a 16-bit- or 32-bit-wide bus as determined by BS16#. The bus is 16 bits wide when BS16# is asserted. When 32 bits wide, memory and I/O spaces are physically addressed as arrays of 32-bit double words. The microprocessor drives the data bus during write bus cycles. The external system hardware drives the data bus during read bus cycles.

Every bus cycle begins with assertion of the address strobe (ADS#). ADS# indicates that the microprocessor has issued a new address and new bus-cycle-definition signals. A bus cycle is defined by four signals: M/IO#, W/R#, D/C#, and LOCK#. M/IO# defines whether a memory or I/O operation is occurring, W/R# defines the cycle as read or write, and D/C# indicates whether a data or control cycle is in effect. LOCK# indicates that the current cycle is a locked bus cycle. Every bus cycle completes when the system hardware returns READY# asserted.

The TI486SXL series microprocessor performs the following bus-cycle types:

- Memory read
- Locked memory read
- Memory write
- Locked memory write
- I/O read (or coprocessor read)
- I/O write (or coprocessor write)
- Interrupt acknowledge (always locked)
- Halt/shutdown

When the microprocessor has no pending bus requests, the bus enters the idle state. There is no encoding of the idle state on the bus-cycle-definition signals; however, the idle state can be identified by the absence of further assertions of ADS# following a completed bus cycle.

Note that all bus diagrams apply to all TI486SXL series microprocessors. The TI486SXL2 clock-doubled feature does not change the external microprocessor bus interface.

4.4.1 Bus Cycles Using Nonpipelined Addressing

The shortest time unit of bus activity is a bus state, commonly called a T state. A bus state is one internal processor clock period in duration (two CLK2 periods in nonclock-doubled mode and one CLK2 period in clock-doubled mode). A complete data transfer occurs during a bus cycle, composed of two or more bus states.

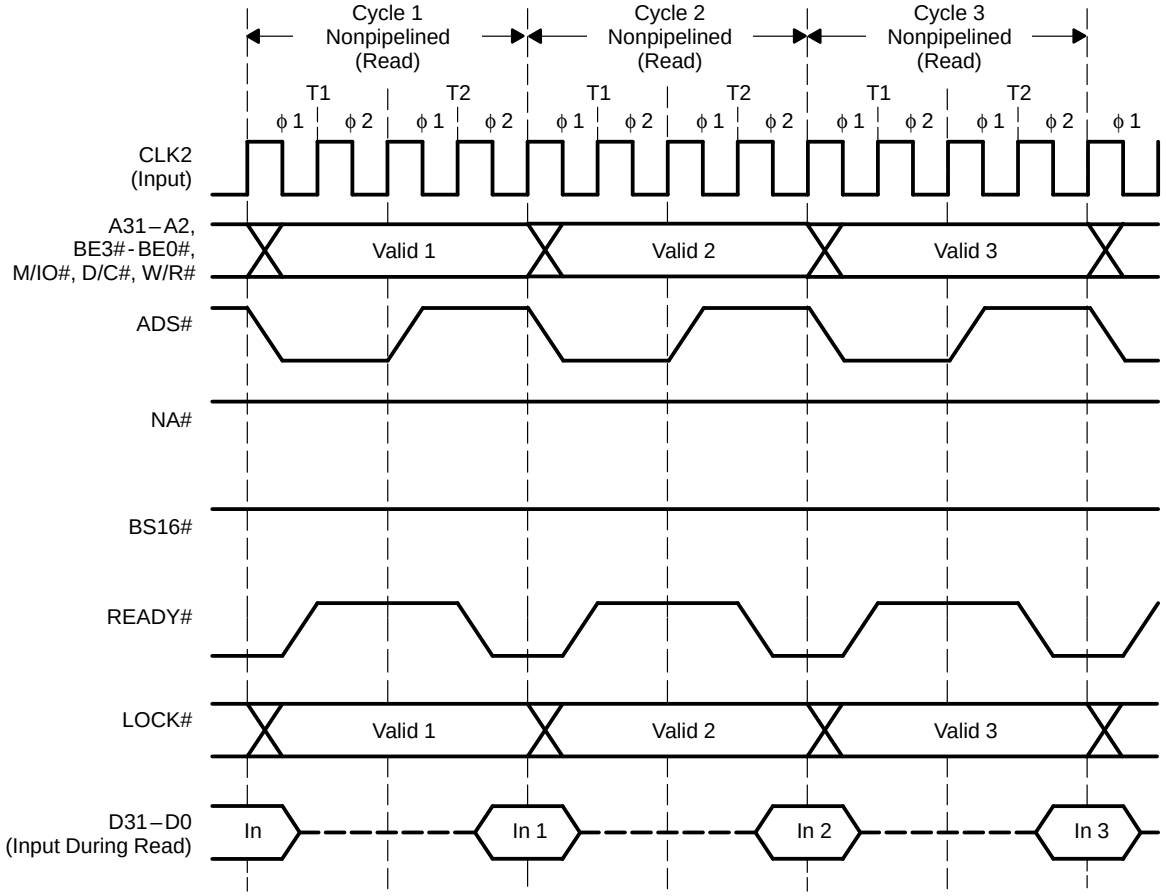
4.4.1.1 Nonpipelined Bus States

The first state of a nonpipelined bus cycle is called T1. During phase one (first CLK2) of T1, the address bus and bus-cycle-definition signals are driven valid and, to signal their availability, address strobe (ADS#) is simultaneously asserted.

The second bus state of a nonpipelined cycle is called T2. T2 terminates a bus cycle with the assertion of the READY# input, and valid data is either read or written depending on the bus-cycle type. The fastest microprocessor bus cycle requires only these two bus states. READY# is ignored at the end of the T1 state.

Three consecutive bus read cycles, each consisting of two bus states, are shown in Figure 4–4.

Figure 4–4. TI486SXL Fastest Nonpipelined Read Cycles



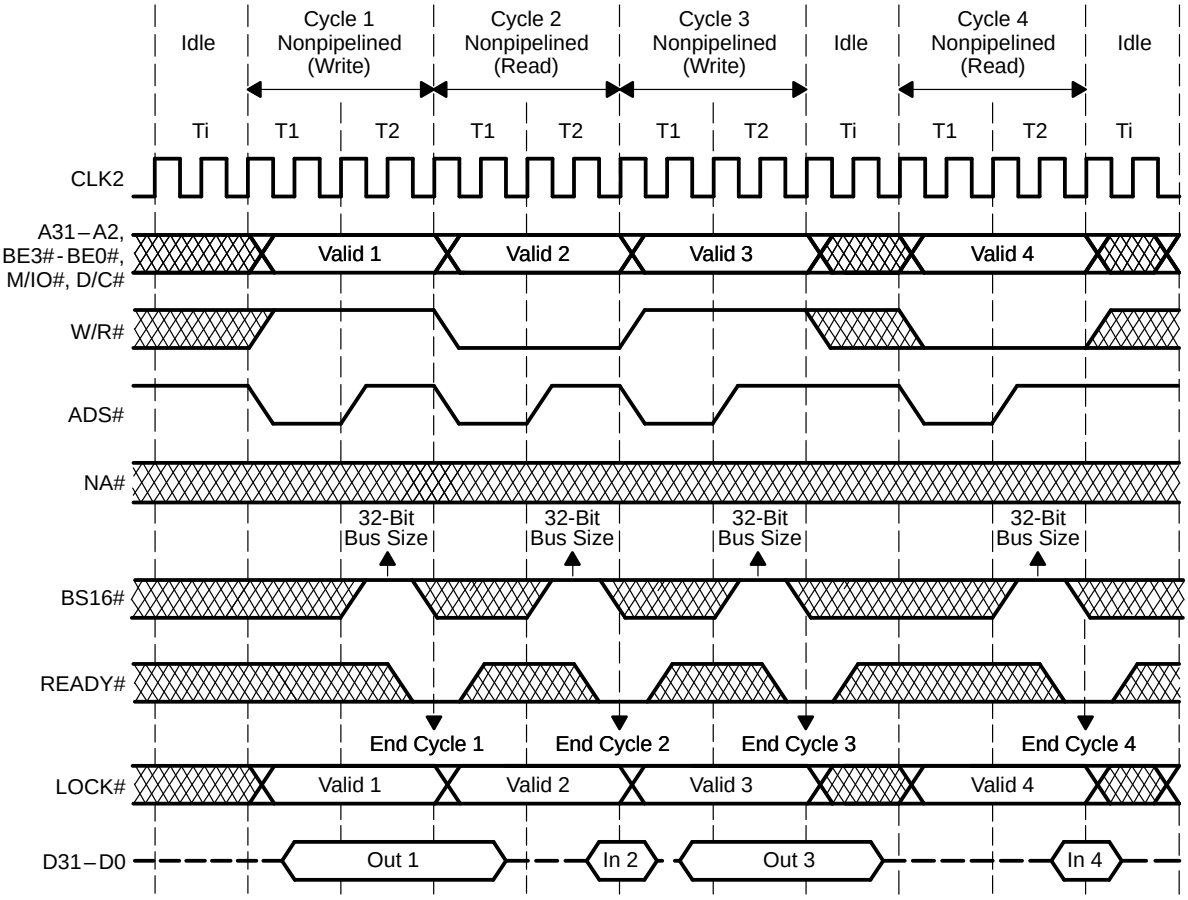
Note: Fastest nonpipelined bus cycles consist of T1 and T2.

4.4.1.2 Nonpipelined Read and Write Cycles

Any bus cycle can be performed with nonpipelined address timing. Figure 4–5 shows a mixture of read and write cycles with nonpipelined address timing. When a read cycle is performed, the microprocessor floats its data bus, and the externally addressed device then drives the data. The microprocessor requires that all data-bus pins be driven to a valid logic state (high or low) at the end of each read cycle, when READY# is asserted. When a read cycle is acknowledged by READY# asserted in the T2 bus state, the microprocessor latches the information present at its data-bus pins and terminates the cycle.

When a write cycle is performed, the data bus is driven by the microprocessor beginning in phase two of T1. When a write cycle is acknowledged, the write data remains valid throughout phase one of the next bus state to provide write-data hold time.

Figure 4–5. TI486SXL Various Nonpipelined Bus Cycles (No Wait States)



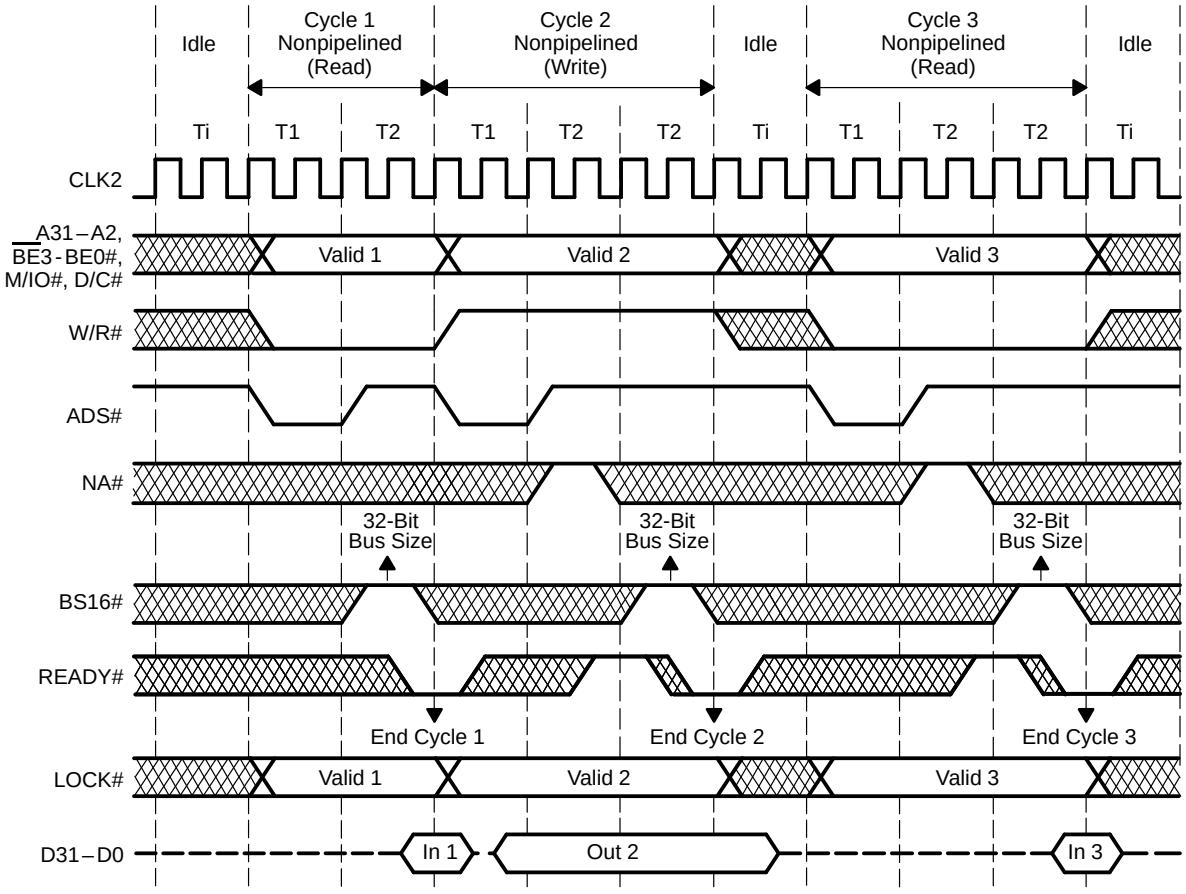
Note: Idle states are introduced arbitrarily.

4.4.1.3 Nonpipelined Wait States

Once a bus cycle begins, it continues until acknowledged by the external system hardware using the READY# input. Acknowledging the bus cycle at the end of the first T2 results in the shortest possible bus cycle, requiring only T1 and T2. However, if READY# is not immediately asserted, T2 states are repeated indefinitely until the READY# input is sampled active. These intermediate T2 states are referred to as wait states. If the external system hardware is not able to receive or deliver data in two bus states, READY# is withheld and adds at least one wait state to the bus cycle. Thus, on an address-by-address basis, the system is able to define how fast a bus cycle completes.

Figure 4–6 illustrates nonpipelined bus cycles with one wait state added to cycles 2 and 3. READY# is sampled inactive at the end of the first T2 state in cycles 2 and 3. Therefore, the T2 state is repeated until READY# is sampled active at the end of the second T2, and the cycle is then terminated. The microprocessor ignores the READY# input at the end of the T1 state.

Figure 4–6. TI486SXL Various Nonpipelined Bus Cycles With Different Numbers of Wait States

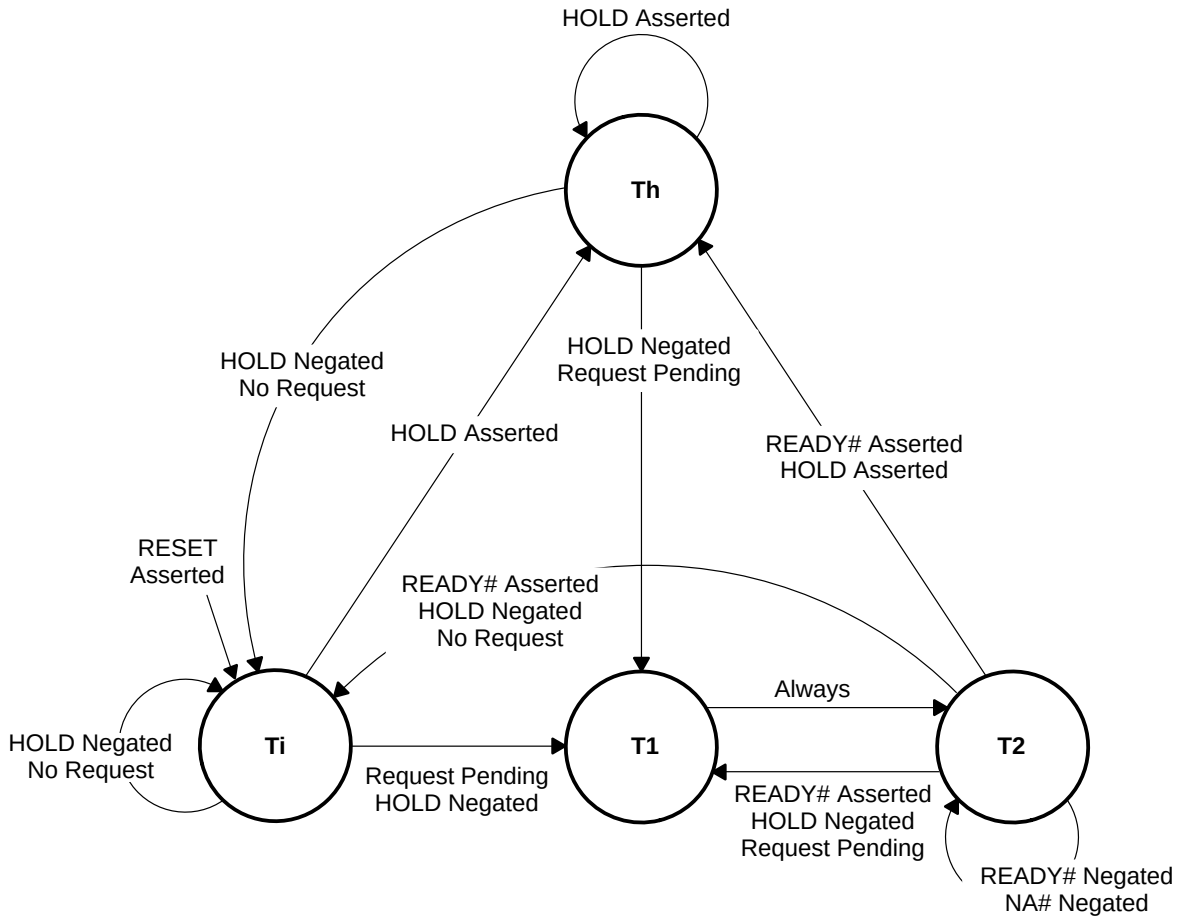


Note: Idle states are introduced arbitrarily.

4.4.1.4 Initiating and Maintaining Nonpipelined Cycles

The bus states and transitions for nonpipelined addressing are illustrated in Figure 4–7. The bus can switch between four possible states: T1, T2, Ti, and Th. Active bus cycles consist of T1 and T2 states, with T2 repeated for wait states. Bus cycles always begin with a single T1 state. T1 is always followed by a T2 state. If a bus cycle is not acknowledged during a given T2 and NA# is inactive, T2 repeats, resulting in a wait state. When a cycle is acknowledged during T2, the following state is T1 of the next bus cycle when a bus request is pending internally. If no internal bus request is pending, the Ti state is entered. If the HOLD input is asserted and the microprocessor is ready to enter the hold-acknowledge state, the Th state is entered.

Figure 4–7. TI486SXL Nonpipelined Bus States



Bus States:
T1 – First clock of a nonpipelined bus cycle (CPU drives new address and asserts ADS#)
T2 – Subsequent clocks of a bus cycle when NA# has not been sampled asserted in the current bus cycle
Ti – Idle state
Th – Hold acknowledge (CPU asserts HLDA)

The fastest bus cycle consists of two states: T1 and T2.

Due to the demultiplexed bus, address pipelining gives the external hardware an additional T state of access time without inserting a wait state. The processor always uses nonpipelined address timing after the reset sequence and following any idle bus state. Pipelined or nonpipelined address timing is then determined on a cycle-by-cycle basis using the NA# input. When address pipelining is not used, the address and bus-cycle definition remain valid during all wait states. When wait states are added and nonpipelined address timing is necessary, NA# should be negated during each T2 state of the bus cycle except the last one.

4.4.2 Bus Cycles Using Pipelined Addressing

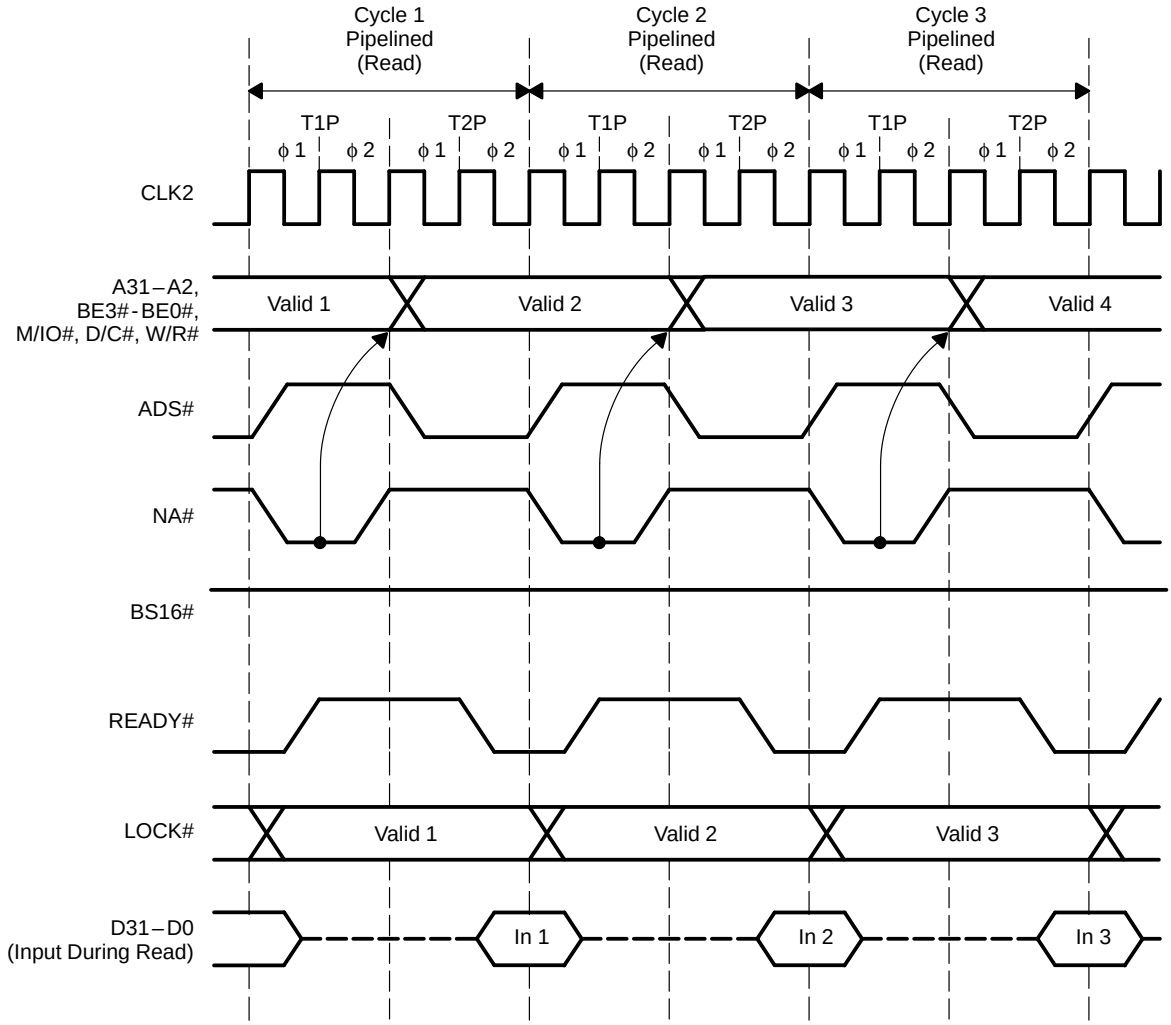
Address pipelining lets the system request the address and bus-cycle definition of the next internally pending bus cycle before the current bus cycle is acknowledged with READY# asserted. If address pipelining is used, the external system hardware has an extra T state of access time to transfer data. Address pipelining is controlled cycle-by-cycle by the state of the NA# input.

4.4.2.1 Pipelined Bus States

Pipelined addressing is always initiated by asserting NA# during a nonpipelined bus cycle. Within the nonpipelined bus cycle, NA# is sampled at the beginning of phase two of each T2 state and is only acknowledged by the microprocessor during wait states. When address pipelining is acknowledged, the address (BE3#–BE0#, and A31–A2) and bus-cycle definition (W/R#, D/C#, and M/IO#) of the next bus cycle are driven before the end of the nonpipelined cycle. The address status output (ADS#) is asserted simultaneously to indicate validity of these signals. Once in effect, address pipelining is maintained in successive bus cycles by continuing to assert NA# during the pipelined bus cycles.

As in nonpipelined bus cycles, the fastest bus cycles using a pipelined address require only two bus states. Figure 4–8 illustrates the fastest read cycles using pipelined address timing. The two bus states for pipelined addressing are T1P and T2P or T1P and T2I. The T1P state is entered following completion of the bus cycle in which the pipelined address and bus-cycle-definition information was made available, and it is the first bus state of every pipelined bus cycle. In other words, the T1P state follows a T2 state if the previous cycle was nonpipelined, and follows a T2P state if the previous cycle was pipelined.

Figure 4–8. TI486SXL Fastest Pipelined Read Cycles



Note: Fastest pipelined bus cycles consist of T1P and T2P.

Within the pipelined bus cycle, NA# is sampled at the beginning of phase two ($\phi 2$) of the T1P state. If the microprocessor has an internally pending bus request and NA# is asserted, the T1P state is followed by a T2P state and the address and bus-cycle definition for the next pending bus request is made available. If no pending bus request exists, the T1P state is followed by a T2I state regardless of the state of NA# and no new address or bus-cycle information is driven.

The pipelined bus cycle is terminated in either the T2P or T2I states with the assertion of the READY# input, and valid data is either input or output depending on the bus-cycle type. READY# is ignored at the end of the T1P state.

4.4.2.2 Pipelined Read and Write Cycles

Any bus cycle can be performed with pipelined address timing. When a read cycle is performed, the microprocessor floats its data bus and the externally addressed device drives the data. When a read cycle is acknowledged by READY# asserted in either the T2P or T2I bus state, the microprocessor latches the information present at its data pins and terminates the cycle.

When a write cycle is performed, the data bus is driven by the microprocessor beginning in phase two ($\phi 2$) of T1P. When a write cycle is acknowledged, the write data remains valid throughout phase one ($\phi 1$) of the next bus state to provide write-data hold time.

4.4.2.3 Pipelined Wait States

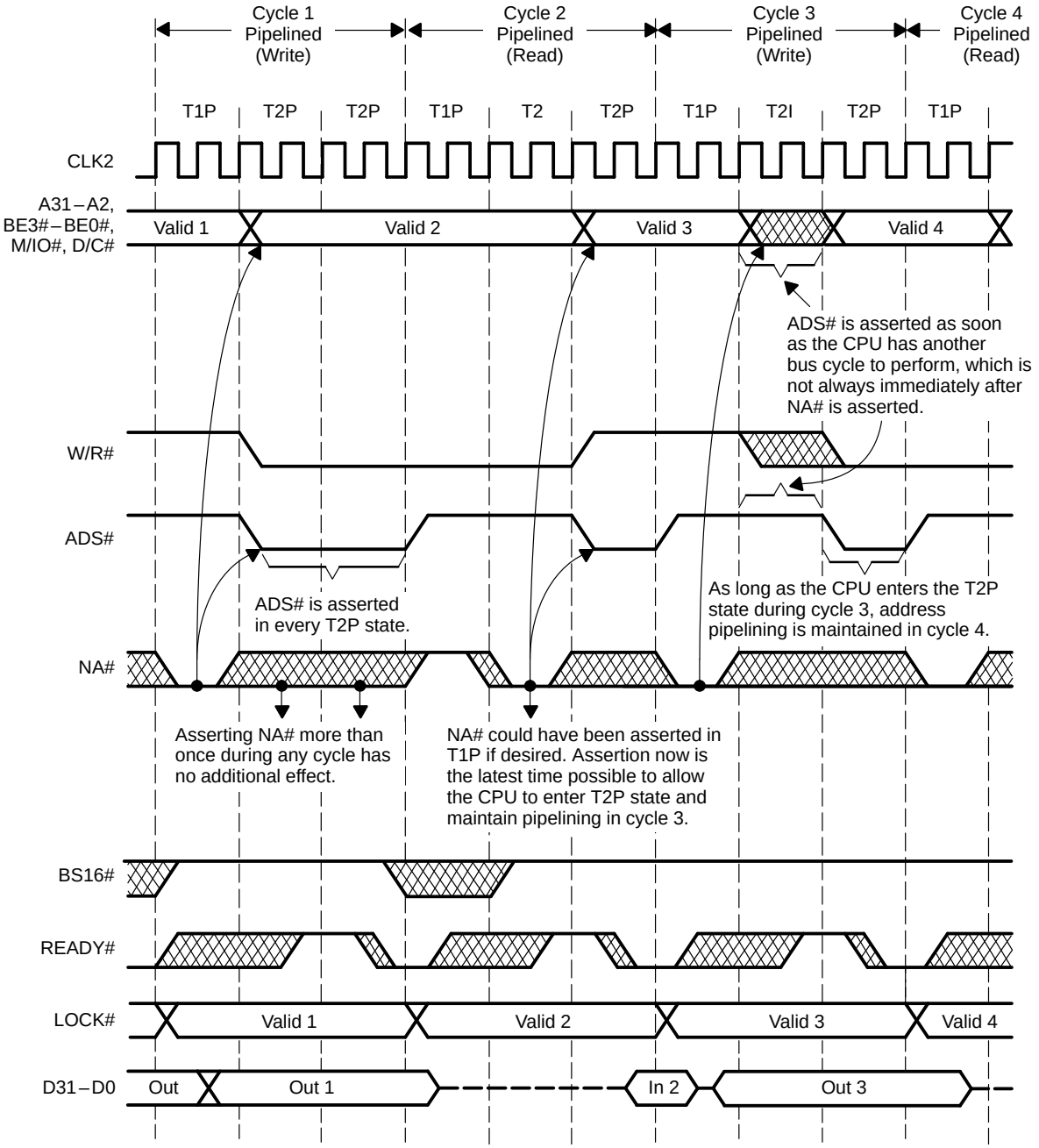
Once a pipelined bus cycle begins, it continues until acknowledged by the external system hardware using the microprocessor READY# input. Acknowledging the bus cycle at the end of the first T2P or T2I state results in the shortest possible pipelined bus cycle. If READY# is not immediately asserted, however, T2P or T2I states are repeated indefinitely until the READY# input is sampled active. Additional T2P or T2I states are referred to as wait states.

Figure 4–9 illustrates pipelined bus cycles with one wait state added to cycles 1 through 3. Cycle 1 is a pipelined cycle with NA# asserted during T1P and a pending bus request. READY# is sampled inactive at the end of the first T2P state in cycle 1. Therefore, the T2P state is repeated until READY# is sampled active at the end of the second T2P, and the cycle is then terminated. The microprocessor ignores the READY# input at the end of the T1P state. ADS#, the address, and the bus-cycle-definition signals for the pending bus cycle are all valid during each of the T2P states. Also, asserting NA more than once during the cycle has no additional effect. Pipelined addressing can only output information for the next bus cycle.

Cycle 2 in Figure 4–9 illustrates a pipelined cycle, with one wait state, where NA# is not asserted until the second bus state in the cycle. In this case, the CPU enters the T2 state following T1P because NA# is not asserted. During the T2 state the microprocessor samples NA# asserted. Because a bus request is pending internally and READY# is not active, the CPU enters the T2P state and asserts ADS#, a valid address, and bus-cycle-definition information for the pending bus cycle. The cycle is then terminated by an active READY# at the end of the T2P state.

Cycle 3 of Figure 4–9 illustrates the case where no internal bus request exists until the last state of a pipelined cycle with wait states. In cycle 3, NA# is asserted in T1P, requesting the next address. Because the CPU does not have an internal bus request pending, the T2I state is entered. However, by the end of the T2I state, a bus request exists. Because READY# is not asserted, a wait state is added. The CPU then enters the T2P state and asserts ADS#, a valid address, and bus-cycle-definition information for the pending bus cycle. As long as the CPU enters the T2P state at some point during the bus cycle, pipelined addressing is maintained. NA# needs to be asserted only once during the bus cycle to request pipelined addressing.

Figure 4–9. TI486SXL Various Pipelined Cycles (One Wait State)



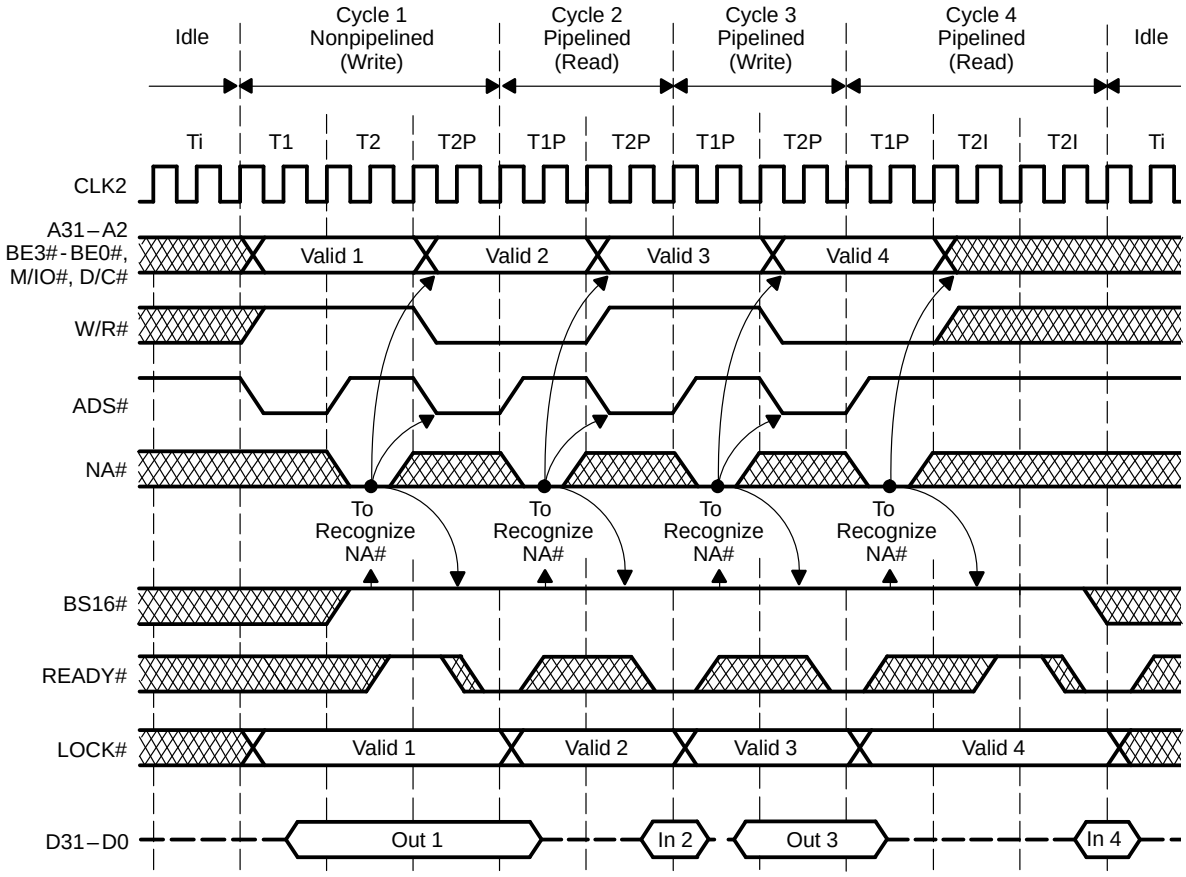
4.4.2.4 Initiating and Maintaining Pipelined Cycles

Pipelined addressing is always initiated by asserting NA# during a nonpipelined bus cycle with at least one wait state. The first bus cycle following reset, an idle bus, or a hold-acknowledge state is always nonpipelined. Therefore, the microprocessor always issues at least one nonpipelined bus cycle following reset, idle, or hold acknowledge before pipelined addressing takes effect.

Once a bus cycle is in progress and the current address has been valid for one entire bus state, the NA# input is sampled at the end of every phase one until the bus cycle is acknowledged. Once NA# is sampled active, the microprocessor is free to drive a new address and bus-cycle definition on the bus as early as the next bus state and as late as the last bus state in the cycle.

Figure 4–10 illustrates the fastest transition possible to pipelined addressing following an idle bus state. In cycle 1, NA# is driven during state T2. Thus, cycle 1 makes the transition to pipelined address timing, since it begins with T1 but ends with T2P. Because the address for cycle 2 is available before cycle 2 begins, cycle 2 is called a pipelined bus cycle, and it begins with a T1P state. cycle 2 begins as soon as READY# assertion terminates cycle 1.

Figure 4–10. TI486SXL Fastest Transition to Pipelined Address Following Bus-Idle State

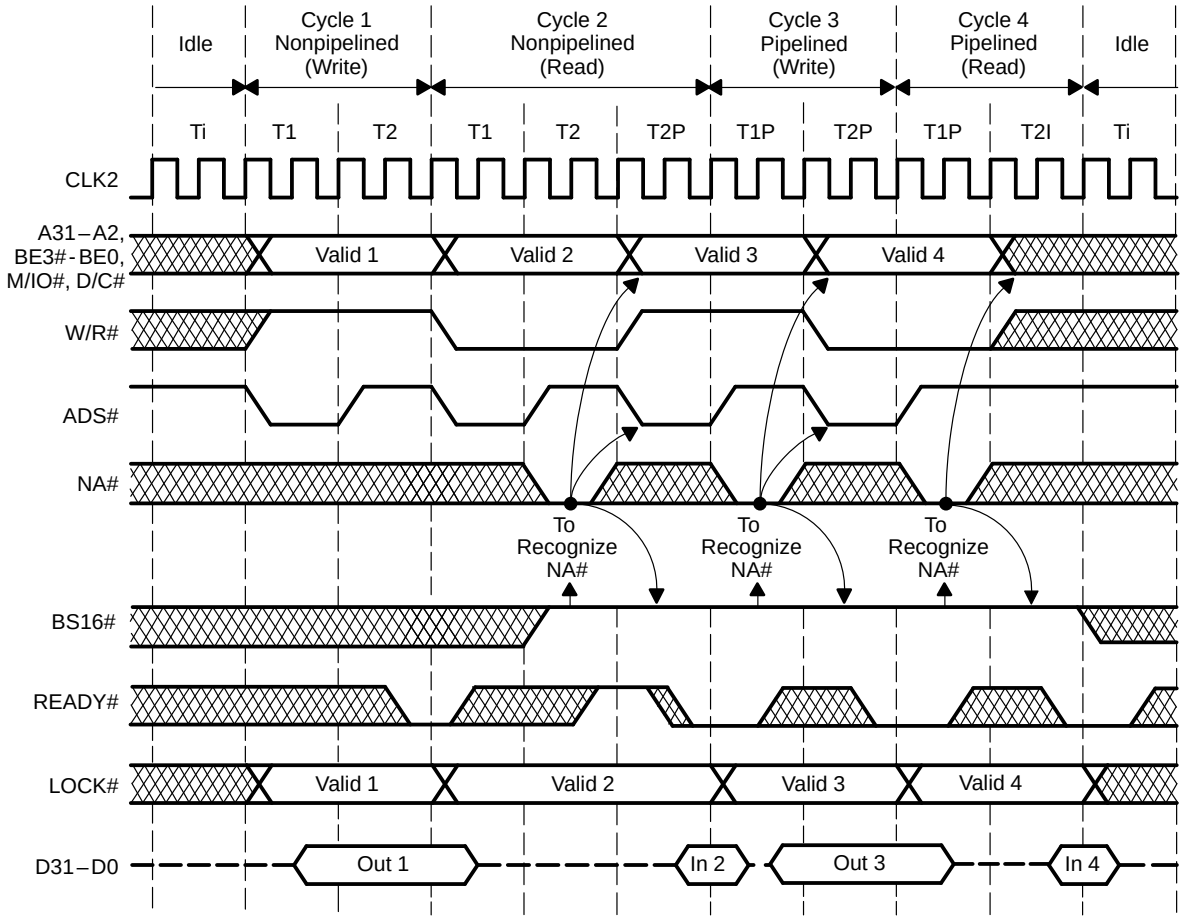


Note: Following any idle bus state (Ti), the address is always nonpipelined and NA# is sampled only during wait states. To start address pipelining after an idle state, a nonpipelined cycle with at least one wait state (cycle 1 above) is required. The pipelined cycles (2, 3, and 4 above) are shown with various numbers of wait states.

Figure 4–11 illustrates the transition to pipelined addressing during a burst of bus cycles. Cycle 2 makes the transition to pipelined addressing. Comparing cycle 2 to cycle 1 of Figure 4–10 (on page 4-30) illustrates that a transition cycle is the same when it occurs and consists of at least T1, T2 (NA# is asserted at that time), and T2P (provided the microprocessor has an internal bus request already pending). T2P states are repeated if wait states are added to the cycle. Cycles 2, 3, and 4 in Figure 4–11 show that once address pipelining is achieved, it can be maintained with two-state bus cycles consisting only of T1P and T2P.

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting NA# and detecting that the microprocessor enters T2P during the current bus cycle. The current bus cycle must end in state T2P for pipelining to be maintained in the next cycle. T2P is identified by the assertion of ADS#. Figure 4–10 and Figure 4–11 each show pipelining ending after cycle 4. This occurs because the microprocessor does not have an internal bus request prior to the acknowledgment of cycle 4.

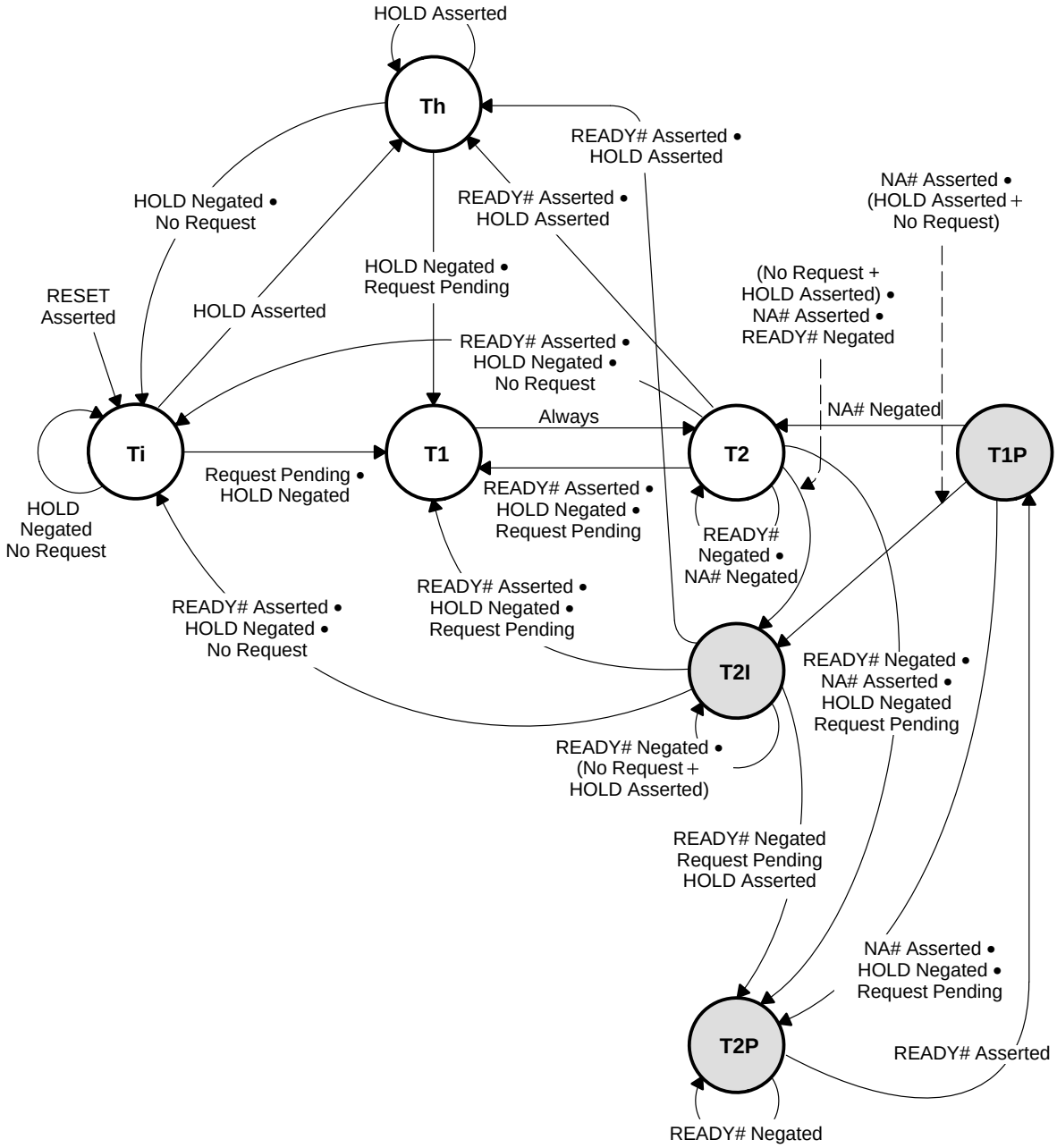
Figure 4–11. TI486SXL Transition to Pipelined Address During Burst of Bus Cycles



Note: Following any idle bus state (Ti), addresses are nonpipelined bus cycles, and NA# is sampled only during wait states. Therefore, to begin address pipelining during a group of nonpipelined bus cycles requires a nonpipelined cycle with at least one wait state (cycle 2 above).

The complete bus-state-transition diagram, including operation with pipelined address, is given in Figure 4–12. This is a superset of the diagram for nonpipelined address. The three additional bus states for pipelined address are shaded.

Figure 4–12. TI486SXL Complete Bus States



Bus States:

T1 – First clock of a nonpipelined bus cycle (CPU drives new address and asserts ADS#)
T2 – Subsequent clocks of a bus cycle when NA# has not been sampled asserted in the current bus cycle
T2I – Subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle but there is not yet an internal bus request pending (CPU does not drive a new address or assert ADS#)
T2P – Subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle and there is an internal bus request pending (CPU drives new address and asserts ADS#)
T1P – First clock of a pipelined bus cycle
Ti – Idle state
Th – Hold acknowledge state (CPU asserts HLDA)

4.4.3 Bus Cycles Using BS16#

Assertion of BS16# during a bus cycle effectively changes the TI486SXL microprocessor 32-bit bus into a 16-bit data bus. Although slower, the 16-bit data bus usually requires less hardware interface circuitry and generally offers greater compatibility with 16-bit devices.

4.4.3.1 Nonpipelined Cycles

With BS16# asserted, all operand transfers physically occur on data bus lines D15–D0. With BS16# asserted during a 32-bit nonpipelined read or write, additional bus cycles are issued by the CPU to transfer the data.

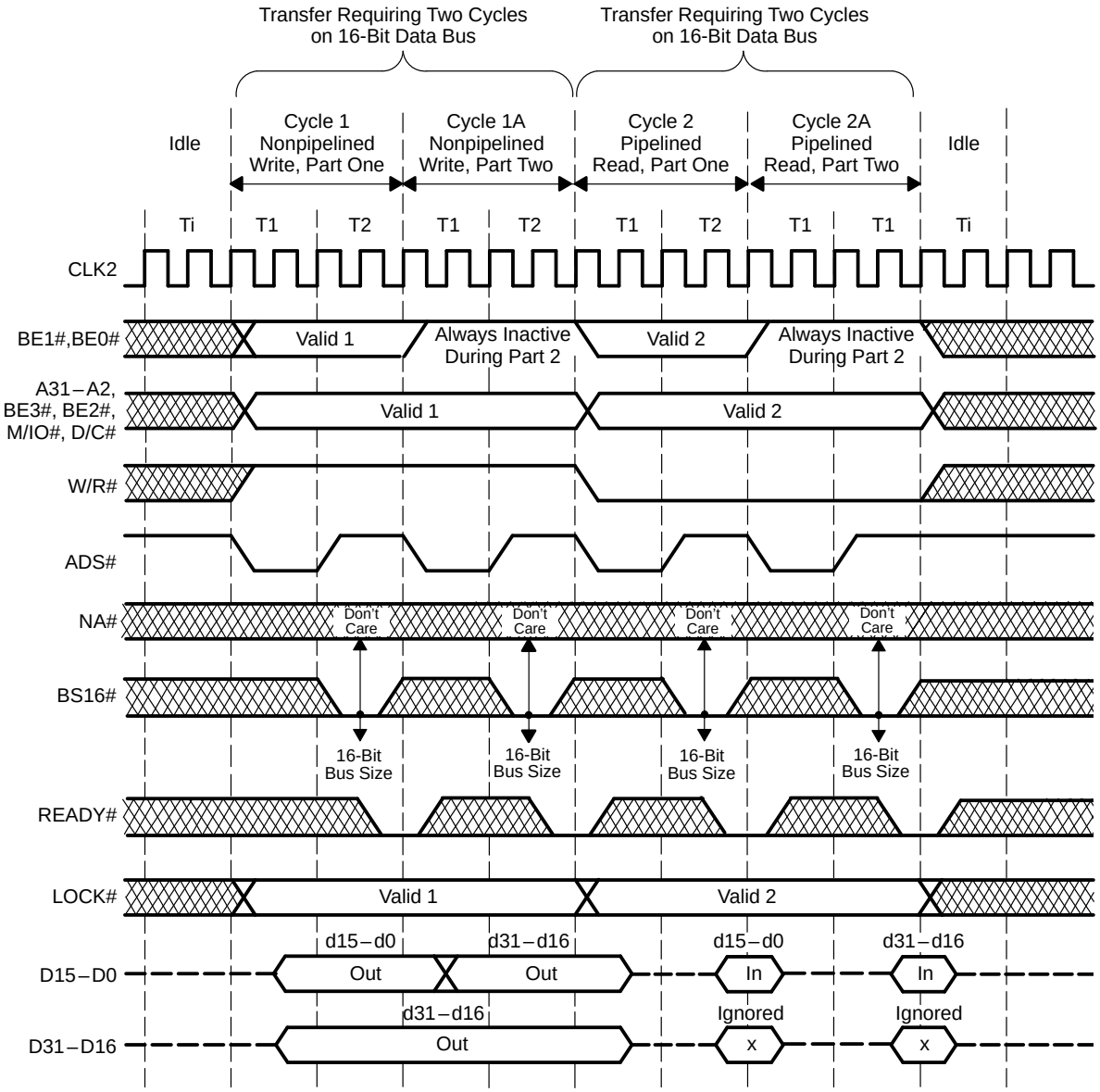
For data reads with only the two upper bytes selected (BE3#/BE2# asserted), data is read from D15–D0.

For data writes with only the two upper bytes selected (BE3#/BE2# asserted), data is duplicated on D15–D0 and no further action is required.

For data reads with all four bytes selected (at least BE1# and BE2# asserted and possibly BE0#/BE3# also asserted), the CPU performs two 16-bit read cycles using data lines D15–D0. Lines D31–D16 are ignored.

Data writes with all four bytes selected (at least BE1# and BE2# asserted and possibly BE0#/BE3# also asserted), the CPU performs two 16-bit write cycles using data lines D15–D0. Bytes 0 and 1 (corresponding to BE0# and BE1#) are sent on the first bus cycle (part one) and bytes 2 and 3 (corresponding to BE2# and BE3#) are sent on the second bus cycle (part two). BE0# and BE1# are always negated during the second 16-bit bus cycle. Figure 4–13 illustrates two nonpipelined bus cycles using BS16#.

Figure 4–13. Nonpipelined Bus Cycles Using BS16#



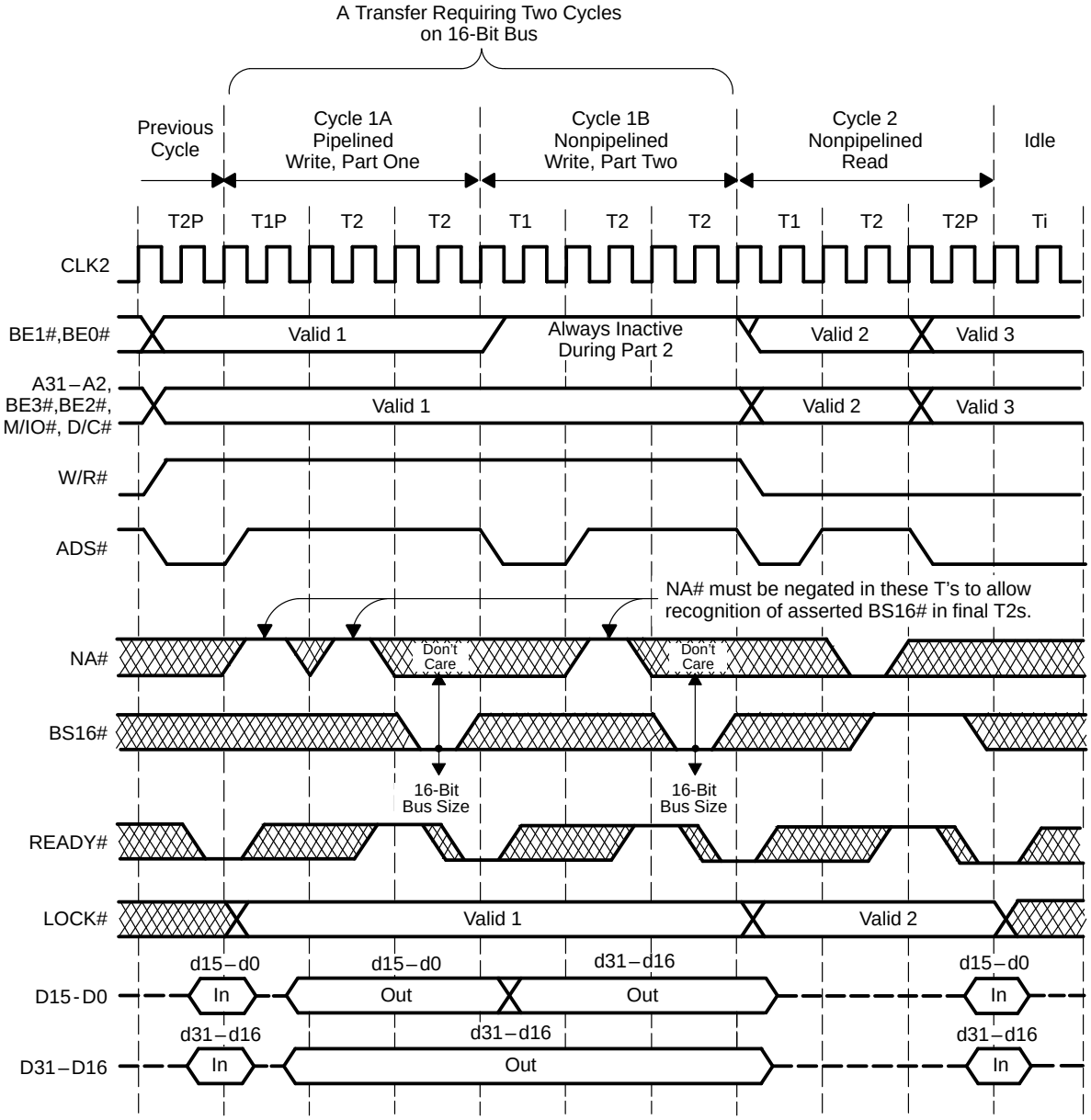
Note: Dn = physical data pin n
dn = logical data bit n

4.4.3.2 Pipelined Cycles

The input signal NA# is a request to the CPU to drive the address, the byte enables, and the bus status signals for the next bus cycle as soon as they become internally available. Pipelining this address allows the system logic to anticipate the next bus-cycle operation.

The CPU cannot acknowledge both address pipelining and BS16# for the same bus cycle. If NA# is already sampled when BS16# is asserted, the data bus remains 32 bits wide. If NA# and BS16# are asserted in the same window, NA# is ignored and BS16# remains effective (the data bus becomes 16 bits wide). Figure 4–14 illustrates the interaction between NA# and BS16#.

Figure 4–14. Pipelining and BS16#



Dn = physical data pin n
dn = logical data bit n
Cycle 1A is pipelined. Cycle 1B cannot be pipelined, but its address can be inferred from cycle 1 to simulate address pipelining externally.

4.4.4 Locked Bus Cycles

When the LOCK# signal is asserted, the TI486SXL series microprocessors do not allow other bus master devices to gain control of the system bus. LOCK# is driven active in response to executing certain instructions with the LOCK prefix. The LOCK prefix allows indivisible read/modify/write operations on memory operands. LOCK# is also active during interrupt-acknowledge cycles.

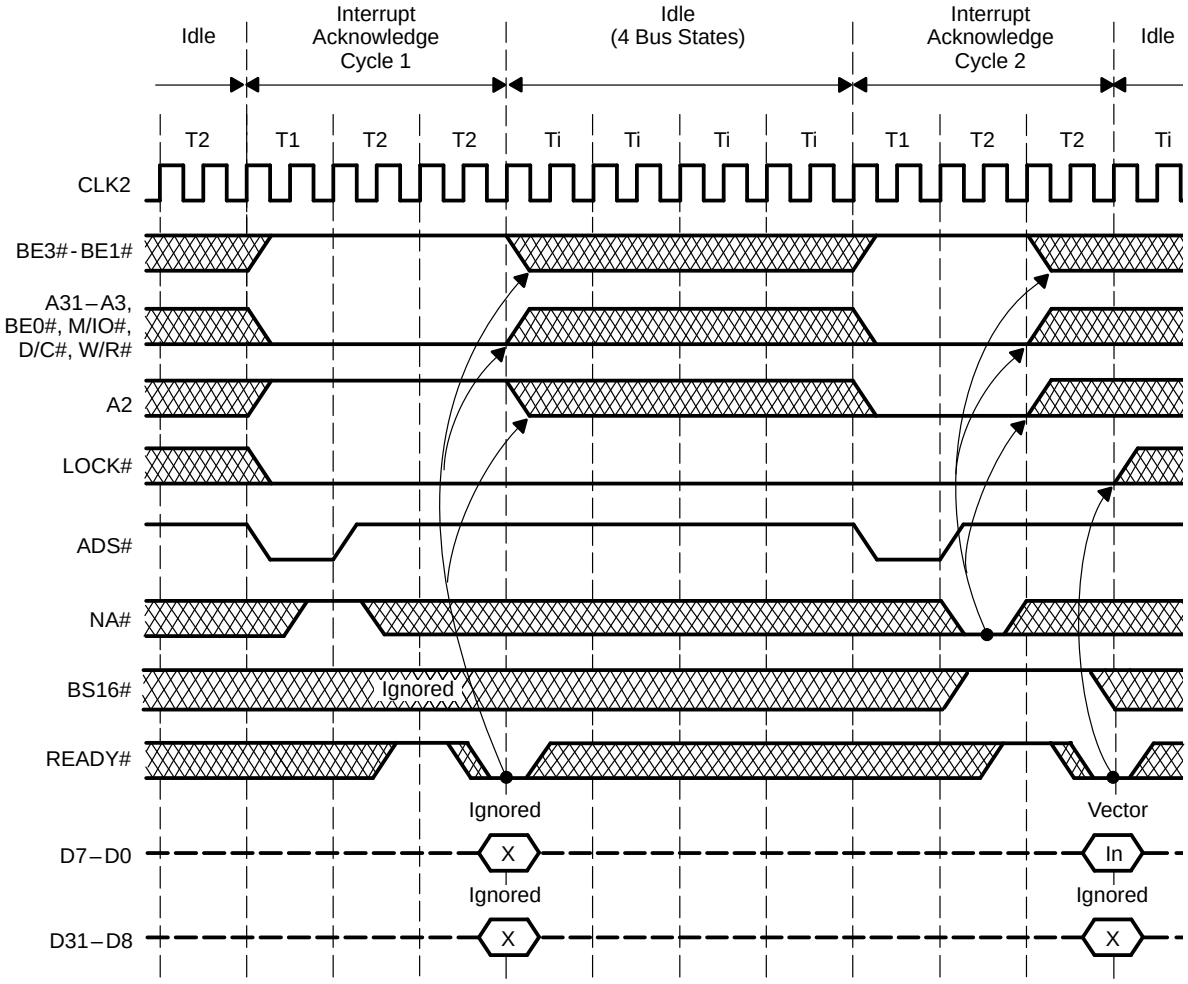
LOCK# is activated on the CLK2 edge that begins the first locked bus cycle and is deactivated when READY# is returned at the end of the last locked bus cycle. When the microprocessor is using nonpipelined addressing, LOCK# is asserted during phase one ($\phi 1$) of T1. When it is using pipelined addressing, LOCK# is driven valid during phase one of T1P.

Figure 4–4, Figure 4–5, Figure 4–6, and Figure 4–13 on pages 4-22, 4-23, 4-24, and 4-34 illustrate LOCK# timing during nonpipelined cycles. Figure 4–8, Figure 4–9, Figure 4–10, Figure 4–11, and Figure 4–14 on pages 4-27, 4-29, 4-30, 4-31 and 4-35 cover the pipelined-address case.

4.4.5 Interrupt-Acknowledge Cycles

The TI486SXL microprocessors are interrupted by an external source via an input request on the INTR input (when interrupts are enabled). The microprocessor responds with two locked interrupt-acknowledge cycles. These bus cycles are similar to read cycles. Each cycle is terminated by READY# sampled active as shown in Figure 4–15.

Figure 4–15. TI486SXL Interrupt-Acknowledge Cycles



Note: Interrupt vector (0–255) is read on D7–D0 at the end of the second interrupt-acknowledge bus cycle. Because each interrupt-acknowledge bus cycle is followed by idle bus states, asserting NA# has no practical effect.

The state of the A2 pin distinguishes the first and second interrupt-acknowledge cycles. The address driven during the first interrupt-acknowledge cycle is 4h (A31–A3 = 0, A2 = 1, BE3#–BE1# = 1, and BE0# = 0). The address driven during the second interrupt-acknowledge cycle is 0h (A31–A2 = 0, BE3#–BE1# = 1, and BE0# = 0).

To assure that the interrupt-acknowledge cycles are executed indivisibly, the LOCK# output is asserted from the beginning of the first interrupt-acknowledge cycle until the end of the second interrupt-acknowledge cycle. In clock-doubled mode, four idle bus states (Ti) are inserted by the microprocessor between the two interrupt-acknowledge cycles. In nonclock-doubled mode, eight idle bus states are inserted.

The interrupt vector is read at the end of the second interrupt cycle. The microprocessor reads the vector from D7–D0 of the data bus. The vector indicates the specific interrupt number (from 0–255) requiring service. Throughout the balance of the two interrupt cycles, D31–D0 float. At the end of the first interrupt-acknowledge cycle, any data presented to the microprocessor is ignored.

4.4.6 Halt and Shutdown Cycles

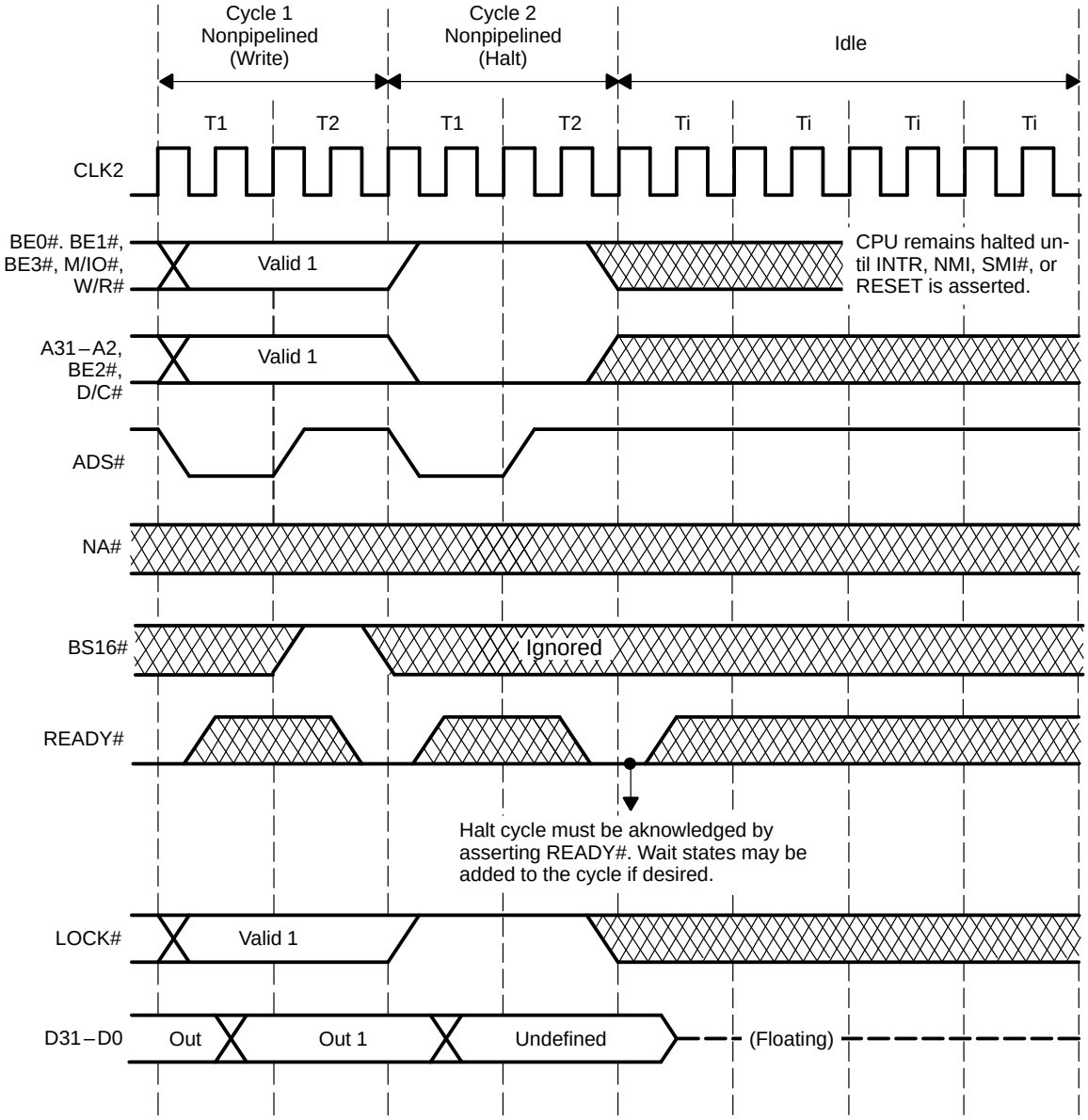
Executing the HLT instruction or detecting a severe error causes the microprocessor to halt operation or shutdown further processing. The microprocessor signals a halt or shutdown through a halt- or shutdown-indication cycle, respectively.

4.4.6.1 Halt Indication Cycle

Executing the HLT instruction causes the microprocessor execution unit to cease operation. The microprocessor signals its entrance into the halt state by performing a halt indication cycle. The halt indication cycle is identified by the state of the bus-cycle-definition signals (M/IO# = 1, D/C# = 0, W/R# = 1, and LOCK# = 1) and an address of 2h (A31–A2 = 0, BE3# = 1, BE2# = 0, and BE1#–BE0# = 1).

The halt indication cycle must be acknowledged by asserting READY#. A halted microprocessor resumes execution when INTR (if interrupts are enabled), NMI, SMI#, or RESET is asserted. Figure 4–16 illustrates a nonpipelined halt cycle.

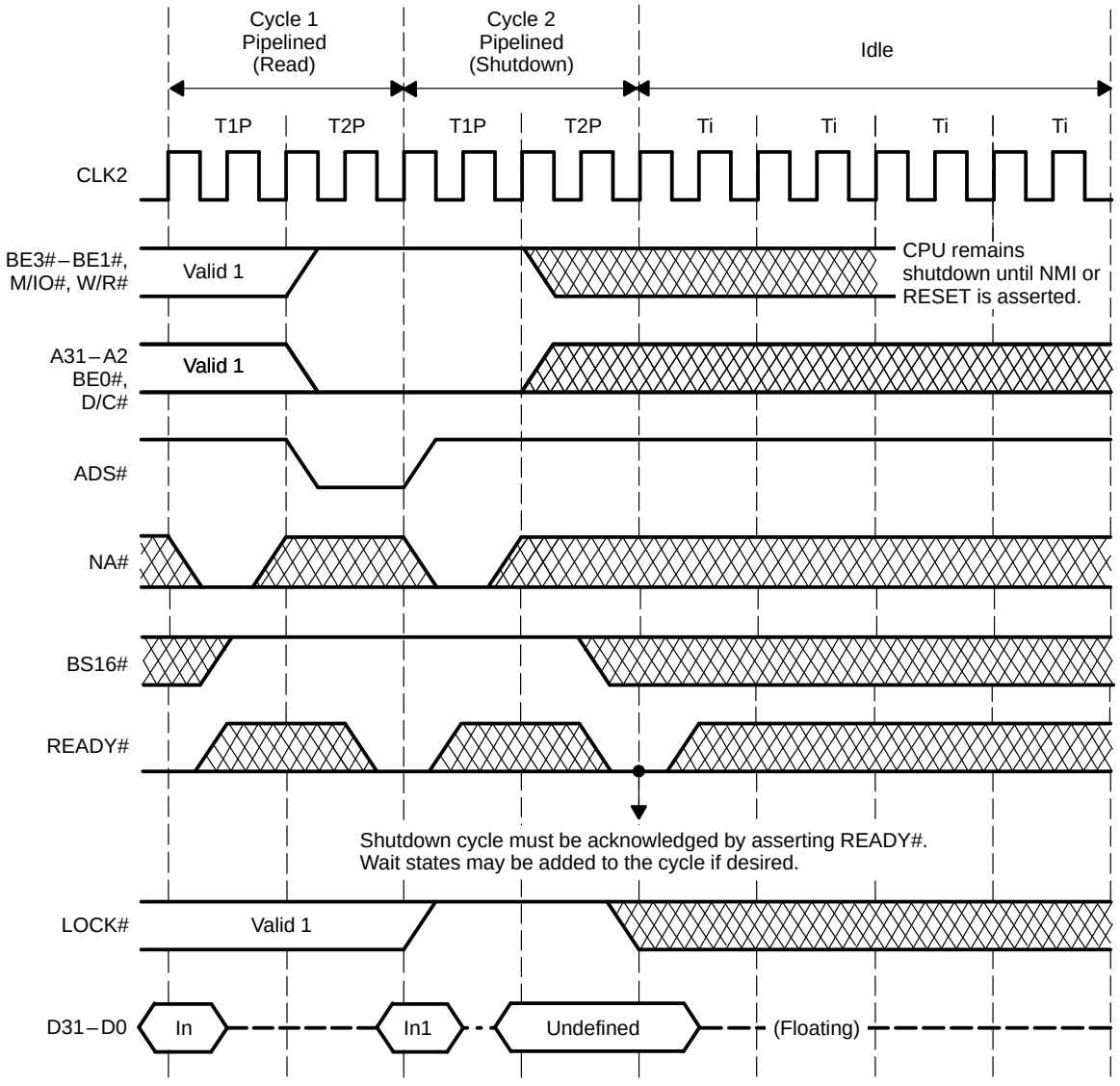
Figure 4–16. TI486SXL Nonpipelined Halt Cycle



4.4.6.2 Shutdown Indication Cycle

Shutdown occurs when the microprocessor detects a severe error that prevents further processing. The TI486SXL series microprocessor shuts down as a result of a protection fault while attempting to process a double fault as well as the conditions referenced in Chapter 2, *Programming Interface*. A shutdown indication cycle is performed, which signals the microprocessor's entrance into the shutdown state. The shutdown indication cycle is identified by the state of the bus-cycle-definition signals (M/IO# = 1, D/C# = 0, W/R# = 1, and LOCK# = 1) and an address of 0h (A31–A2 = 0, BE3#–BE1# = 1, and BE0# = 0). The shutdown indication cycle must be acknowledged by asserting READY#. A shutdown microprocessor resumes execution when NMI or RESET is asserted. Figure 4–17 illustrates a shutdown cycle using pipelined addressing.

Figure 4–17. TI486SXL Pipelined Shutdown Cycle



4.4.7 Internal Cache Interface

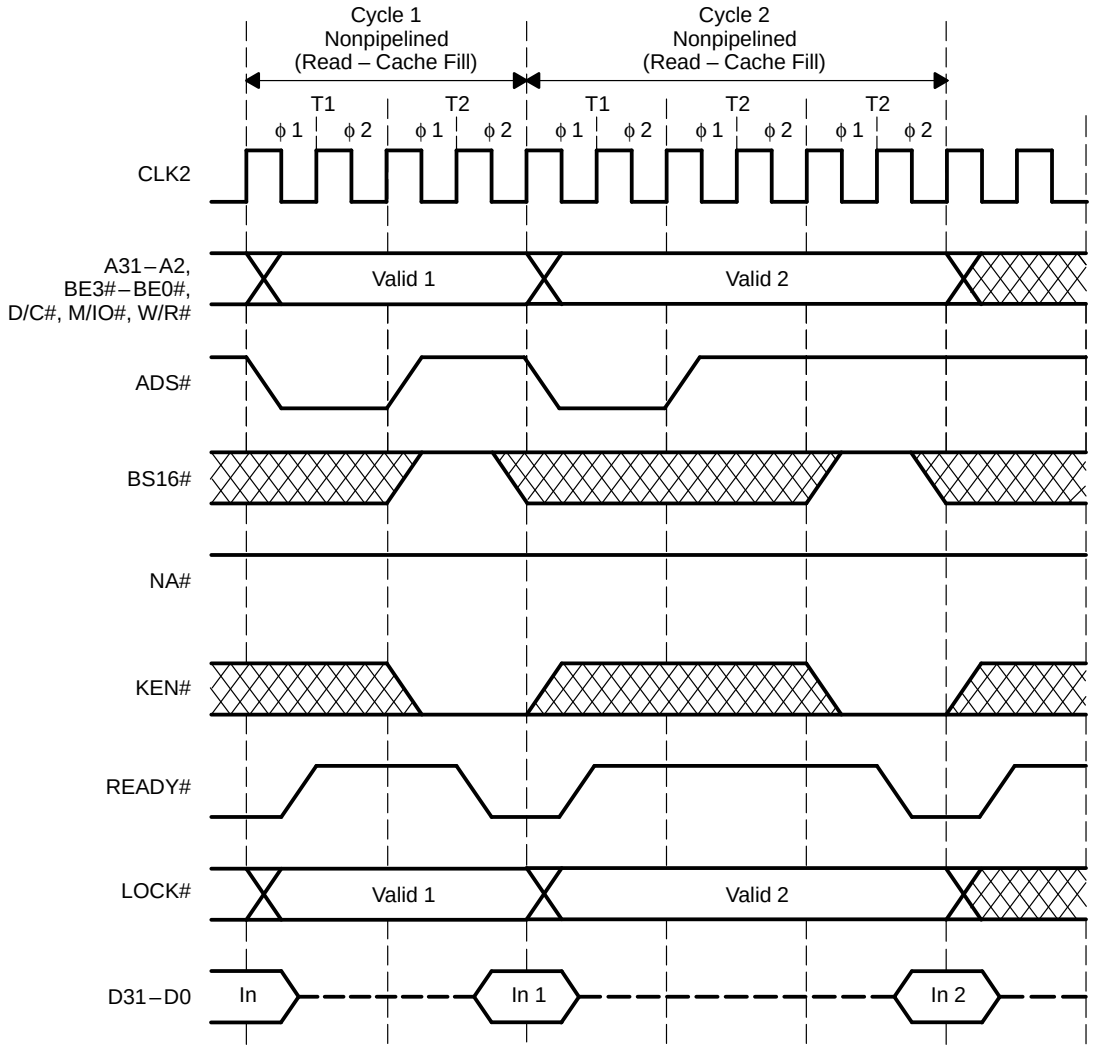
The TI486SXL cache is an 8K-byte write-through unified instruction/data cache with lines that are allocated only during memory read cycles. The cache is configured as two-way set associative. The cache organization consists of 1024 sets, each containing two lines of four bytes each.

4.4.7.1 Cache Fills

Any unlocked memory read cycle can be cached by the TI486SXL series microprocessor. The microprocessor does not automatically cache accesses to memory addresses specified by the Noncacheable-Region registers. Additionally, the KEN# input can be used to enable caching of memory accesses on a cycle-by-cycle basis. The microprocessor acknowledges the KEN# input only if the KEN enable bit is set in the CCR0 Configuration register.

As shown in Figure 4–18, the microprocessor samples the KEN# input one CLK2 period before READY# is sampled active. If KEN# is asserted and the current address is cacheable, the microprocessor fills two bytes of a line in the cache with the data present on the data bus pins.

Figure 4–18. Nonpipelined Cache Fills Using KEN#



As shown in Figure 4–19 and Figure 4–20 on page 4-43, the microprocessor samples the KEN# input one CLK2 period before READY# is sampled active. If KEN# is asserted and the current address is set as cacheable, the microprocessor fills two bytes of a line in the cache with the data present on the data bus pins. The states of BE3#–BE0# are ignored if KEN# is asserted for the cycle.

Figure 4–19. Nonpipelined Cache Fills Using KEN# and BS16#

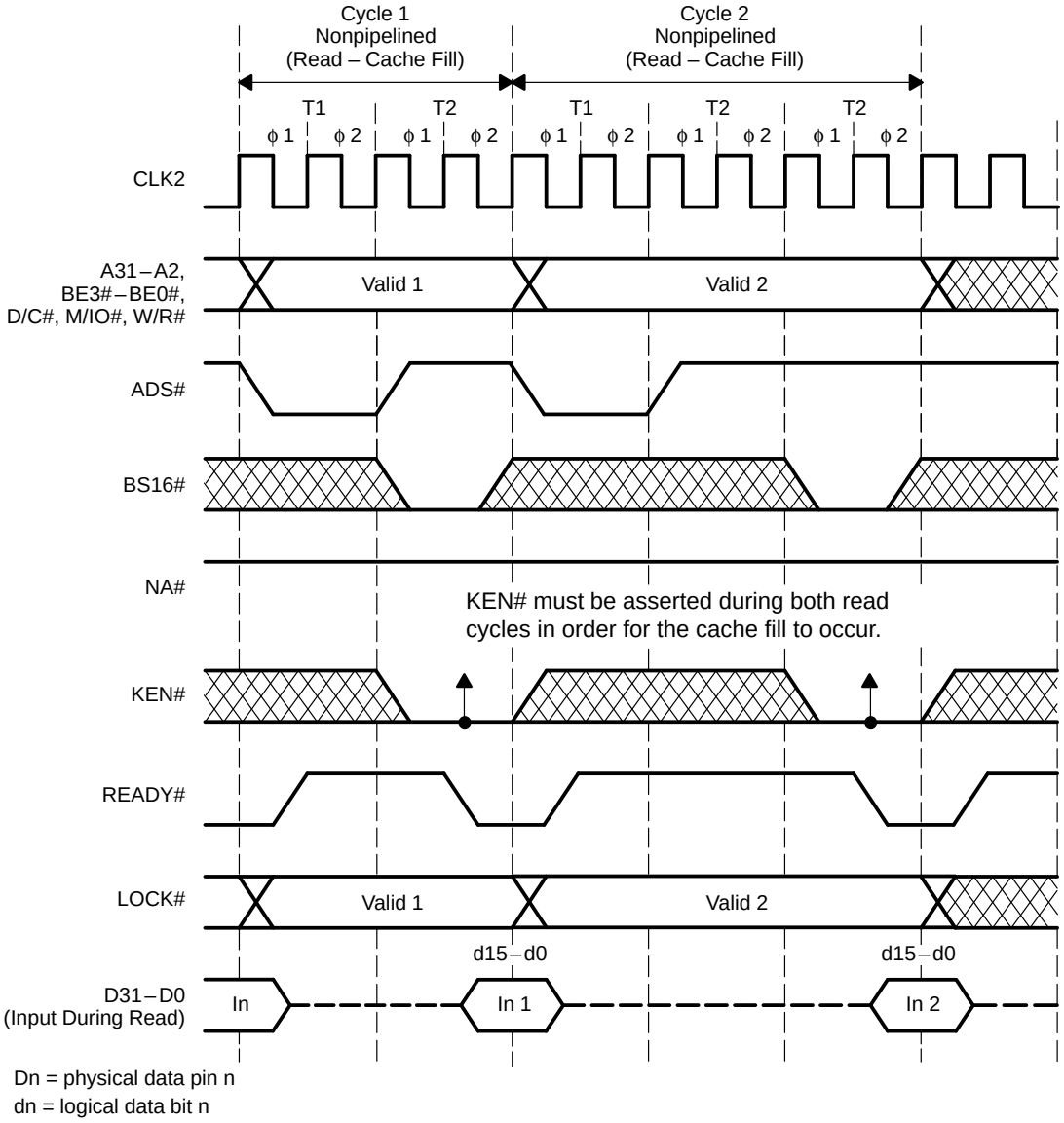
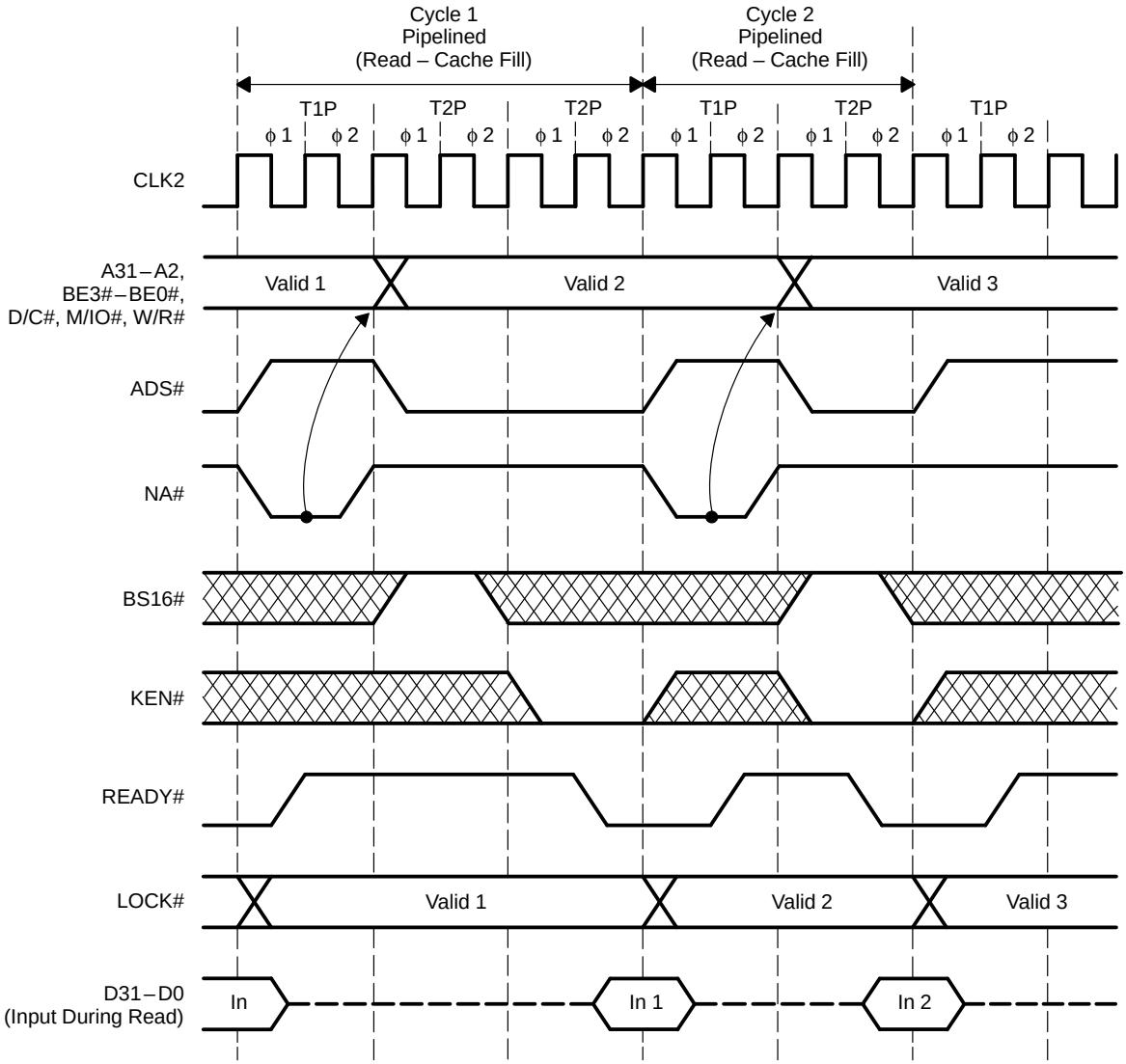


Figure 4–20. Pipelined Cache Fills Using KEN#



4.4.7.2 Flushing the Cache

To maintain cache coherency with external memory, the TI486SXL series microprocessors cache contents should be invalidated when previously cached data is modified in external memory by another bus master. The microprocessor invalidates the internal cache contents during execution of the INVD and WBINVD instructions following assertion of:

- HLDA if the BARB bit is set in the CCR0 Configuration register
- FLUSH# if the FLUSH bit is set in CCR0

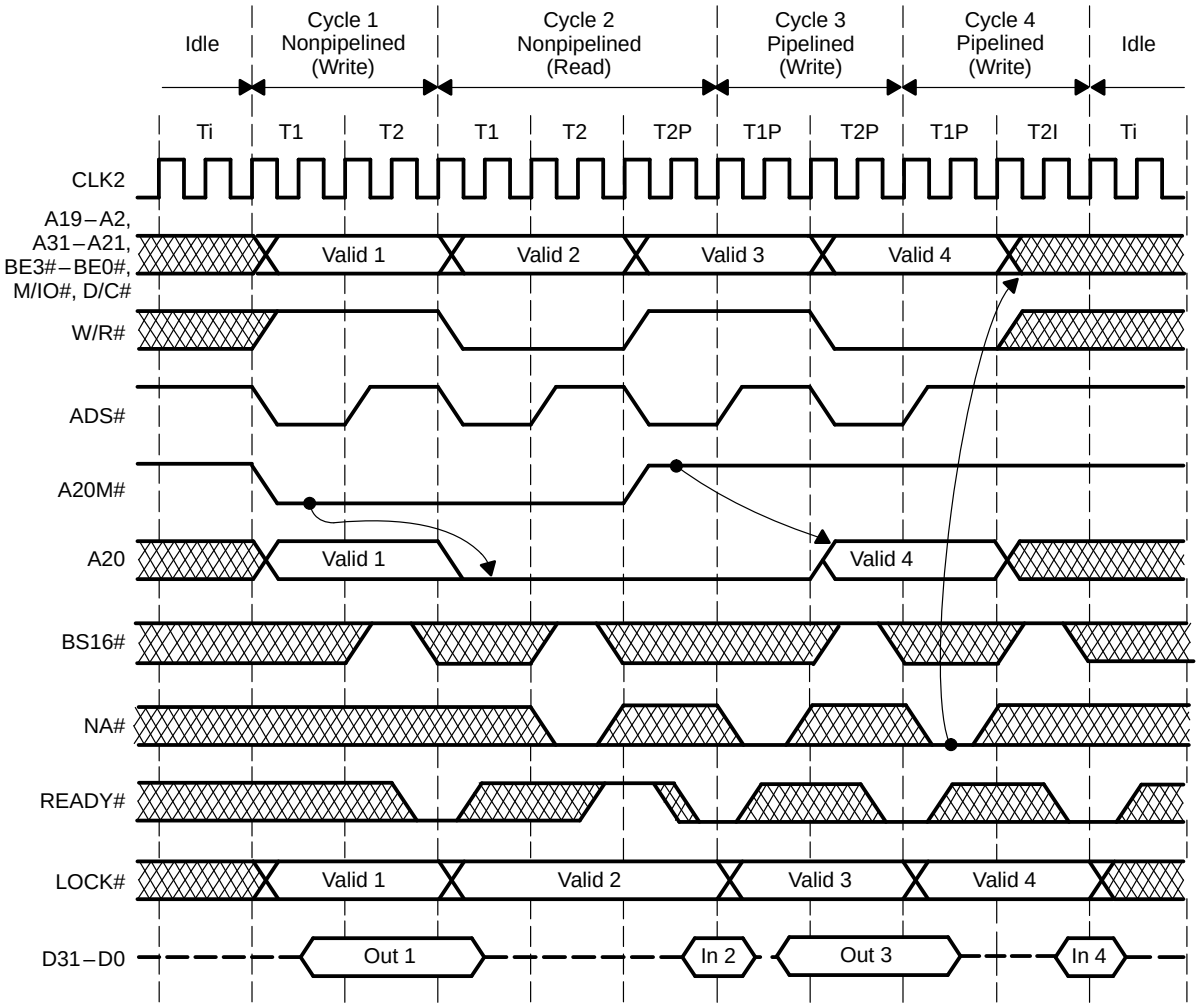
The microprocessor samples the FLUSH# input on the rising edge of CLK2 corresponding to the beginning of phase two ($\phi 2$) of the internal processor clock. If FLUSH# is asserted, the microprocessor invalidates the entire contents of the internal cache. The actual point in time when the cache is invalidated depends upon the internal state of the execution pipeline. FLUSH# must be asserted for at least two CLK2 periods and must meet specified setup and hold times to be recognized on a specific CLK2 edge.

4.4.8 Address Bit-20 Masking

The TI486SXL series microprocessor can be forced to provide 8086 1M-byte address wraparound compatibility by setting the A20M bit in the CCR0 Configuration register and asserting the A20M# input. When the A20M# is asserted, the 20th bit in the address to both the internal cache and the external bus pin is masked (zeroed).

As shown in Figure 4–21, the microprocessor samples the A20M# input on the rising edge of CLK2 corresponding to the beginning of phase 2 ($\phi 2$) of the internal processor clock. If A20M# is asserted and paging is not enabled, the microprocessor masks the A20 signal internally starting with the next cache access and externally starting with the next bus cycle. If paging is enabled, the A20 signal is not masked regardless of the state of A20M#. The A20 signal remains masked until the access following detection of an inactive state on the A20M# pin. A20M# must be asserted for a minimum of two CLK2 periods and must meet specified setup and hold times to be recognized on a specific CLK2 edge.

Figure 4–21. TI486SXL Masking A20 Using A20M# During Burst of Bus Cycles



An alternative to using the A20M# pin is to set the NC0 bit in the CCR0 Configuration register. When the NC0 bit is set, the microprocessor does not automatically cache accesses to the first 64K bytes and to 1M byte + 64K bytes. This prevents data within the wraparound memory area from residing in the internal cache and eliminates the need for masking address A20 to the internal cache.

4.4.9 Hold Acknowledge State

The hold-acknowledge state lets an external device in a TI486SXL microprocessor system acquire the system bus while the microprocessor is held in an inactive bus state. This allows external bus masters to take control of the microprocessor bus and directly access system hardware in a shared manner. The microprocessor continues to execute instructions out of the internal cache (if enabled) until a system bus cycle is required.

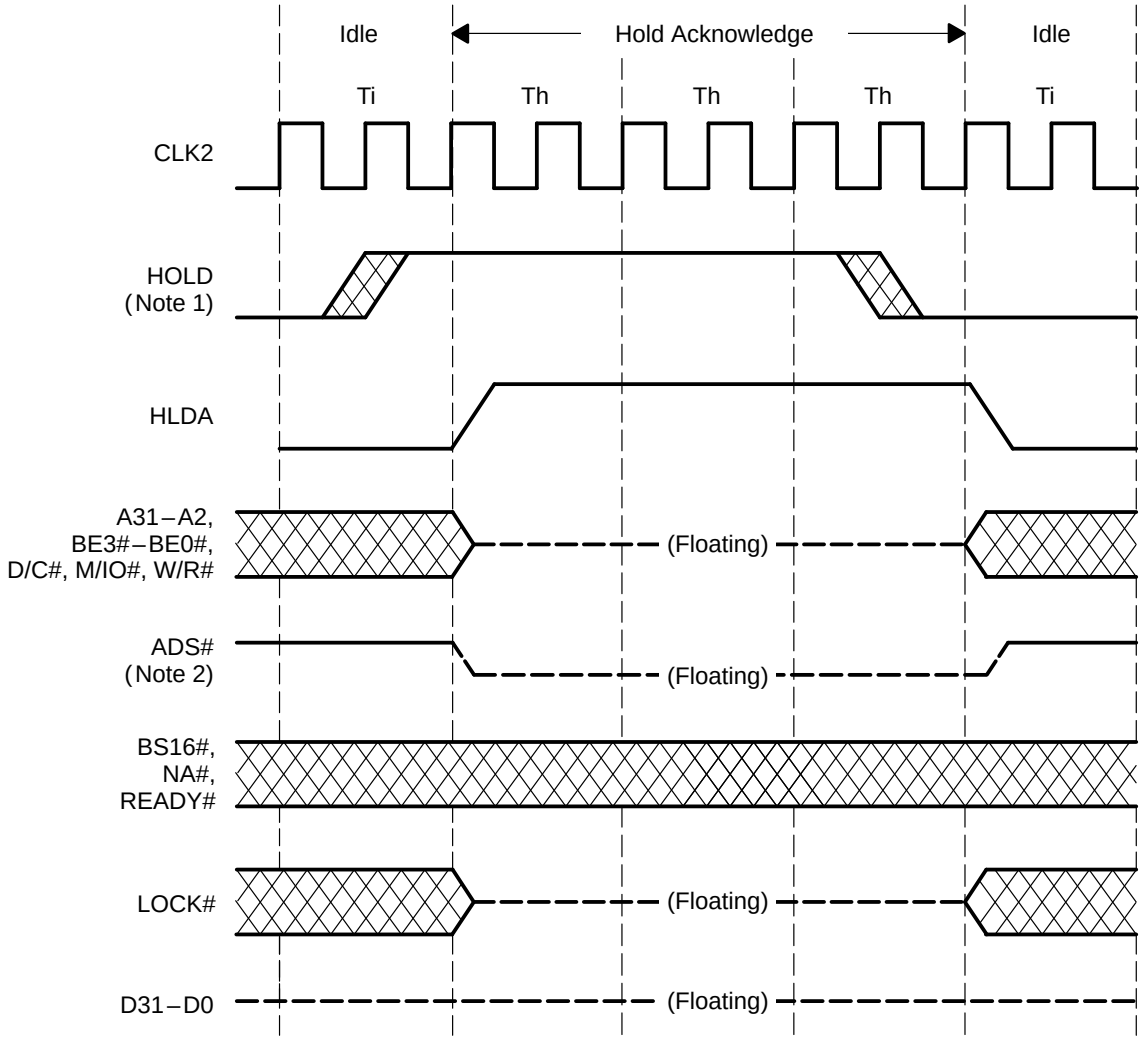
The hold-acknowledge state (Th) is entered in response to assertion of the HOLD input. In the hold-acknowledge state, the microprocessor floats all output and bidirectional signals, except for HLDA and SUSPA#. HLDA is asserted as long as the microprocessor remains in the hold-acknowledge state. All inputs except HOLD, FLUSH#, SUSP# and RESET are ignored.

State Th can be entered directly from a bus-idle state, as in Figure 4–22, or after the completion of the current physical bus cycle if the LOCK signal is not asserted, as in Figure 4–23 and Figure 4–24. The CPU samples the HOLD input on the rising edge of CLK2 corresponding to the beginning of phase one ($\phi 1$) of the internal processor clock. HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold requirements are met in every bus state.

The hold-acknowledge state is exited in response to the HOLD input being negated. The next bus start is an idle state (Ti) if no bus request is pending, as in Figure 4–22. If an internal bus request is pending, as in Figure 4–23 and Figure 4–24, the next bus state is T1. Th is also exited in response to RESET being asserted. If HOLD remains asserted when RESET goes inactive, the microprocessor enters the hold-acknowledge state before performing any bus cycles provided HOLD is still asserted when the CPU is ready to perform its first bus cycle.

If a rising edge occurs on the edge-triggered NMI input while in Th state, the event is remembered as a nonmaskable interrupt 2 and is serviced when the state is exited.

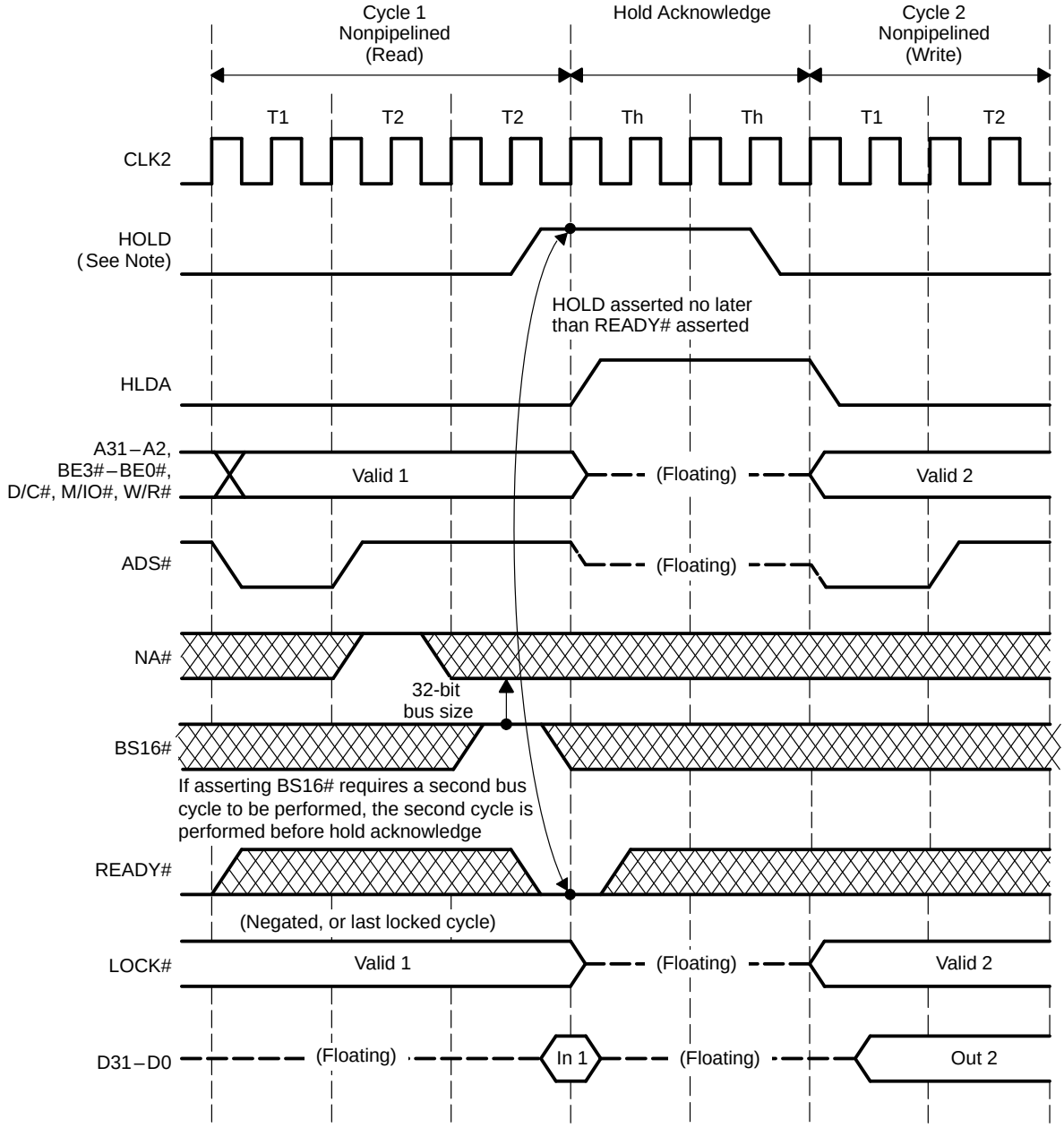
Figure 4–22. TI486SXL Requesting Hold From Bus-Idle State



Notes: 1) HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold requirements are met in every bus state. Violating setup or hold requirements results in incorrect operation.

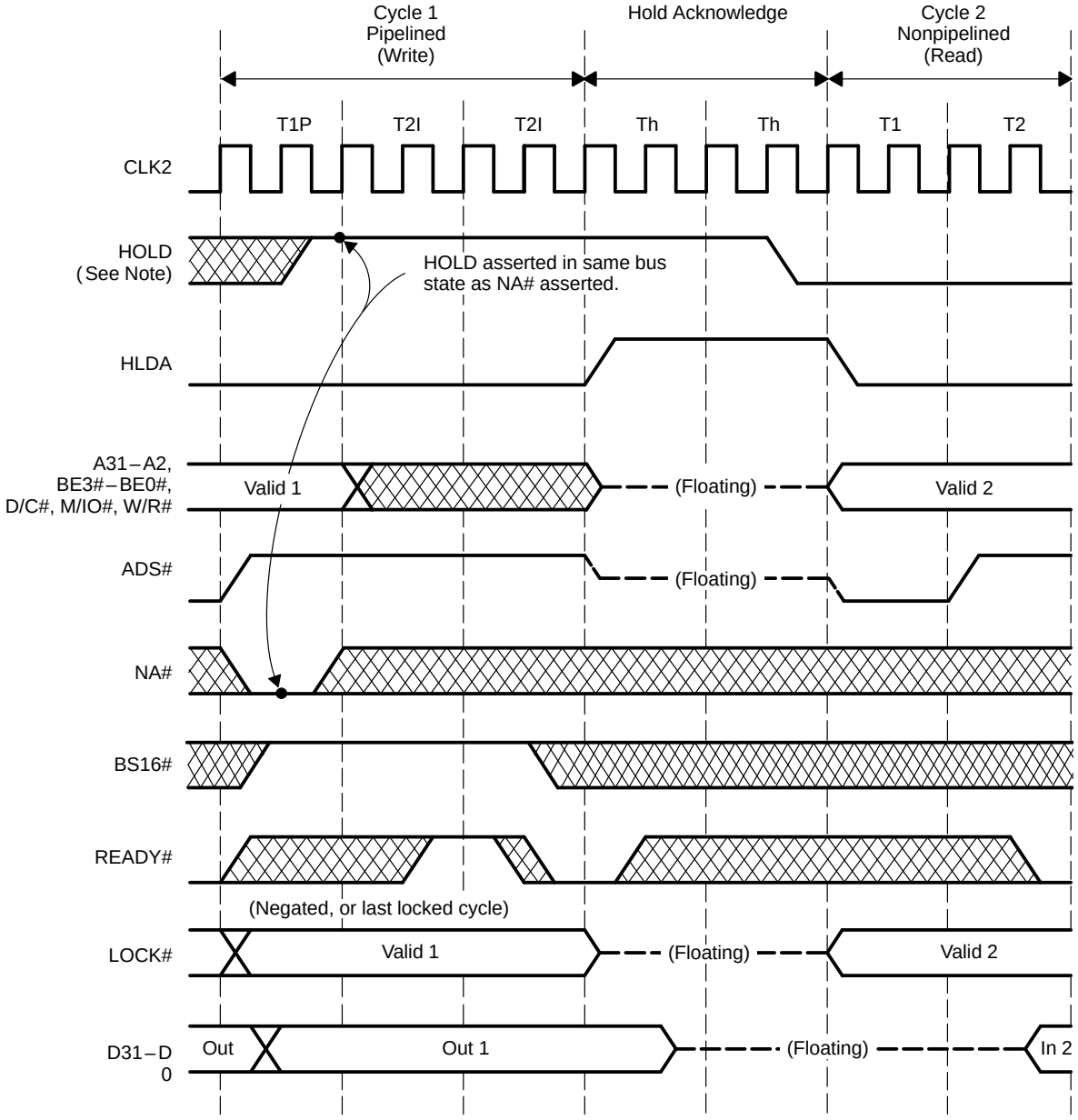
2) For maximum design flexibility, the CPU has no internal pullup resistors on its outputs. External pullups may be required on ADS# and other outputs to keep them negated during the hold-acknowledge period.

Figure 4–23. TI486SXL Requesting Hold From Active Nonpipelined Bus



Note: HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold requirements are met in every bus state. Violating setup or hold requirements results in incorrect operation.

Figure 4–24. TI486SXL Requesting Hold from Active Pipelined Bus



Note: HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold requirements are met in every bus state. Violating setup or hold requirements will result in incorrect operation.

4.4.10 Coprocessor Interface

The data-bus, address-bus, and bus-cycle-definition signals, and the coprocessor interface signals (PEREQ, BUSY#, and ERROR#), control communication between the TI486SXL series microprocessor and a coprocessor. The microprocessor decodes coprocessor or ESC opcodes and transfers the opcode and operands to the coprocessor via I/O port accesses. Address 8000 00F8h functions as the control-port address, and 8000 00FCh is used for operand transfers.

Coprocessor cycles can be read or write and can be nonpipelined or pipelined. Coprocessor cycles must be terminated by READY# and, as with any other bus cycle, can be terminated as early as the second bus state of the cycle.

BUSY#, ERROR#, and PEREQ are asynchronous level-sensitive inputs that synchronize CPU and coprocessor operation. All three signals are sampled at the beginning of phase one ($\phi 1$) and must meet specified setup and hold times to be recognized at a given CLK2 edge.

4.4.11 SMM Interface

System Management Mode (SMM) uses two TI486SXL microprocessor pins, SMI# and SMADS#. The bidirectional SMI# pin is a nonmaskable interrupt that is a higher priority than the NMI input. SMI# must be active for at least four CLK2 periods to be recognized by the microprocessor. Once the microprocessor recognizes the active SMI# input, the CPU drives the SMI# pin low for the duration of the SMI service routine.

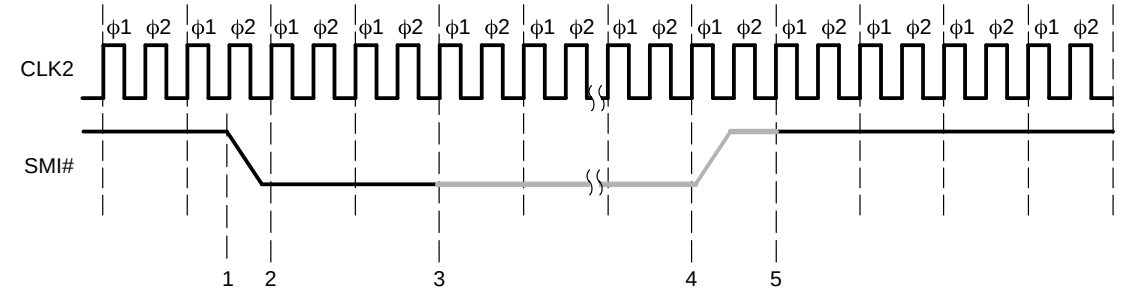
The SMADS# pin outputs the SMM address strobe that indicates an SMM memory bus cycle is in progress and a valid SMM address is on the address bus. The SMADS# functional timing, output delay times, and float delay times are identical to the main memory address strobe (ADS#) timing.

4.4.11.1 SMI Handshake

The functional timing for the SMI# interrupt is shown in Figure 4–25. Five significant events take place during an SMI# handshake:

1) The SMI# input pin is driven active (low) by the system logic.
2) The CPU samples SMI# active on the rising edge of CLK2 phase one ($\phi 1$).
3) Four CLK2 edges after sampling the SMI# active, the CPU switches the SMI# pin to an output and drives SMI# low.
4) Following execution of the RSM instruction, the CPU drives the SMI# pin high for two CLK2 edges indicating completion of the SMI service routine.
5) 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 the inactive (high) level after the pin has been changed to an input.

Figure 4–25. TI486SXL SMI# Timing

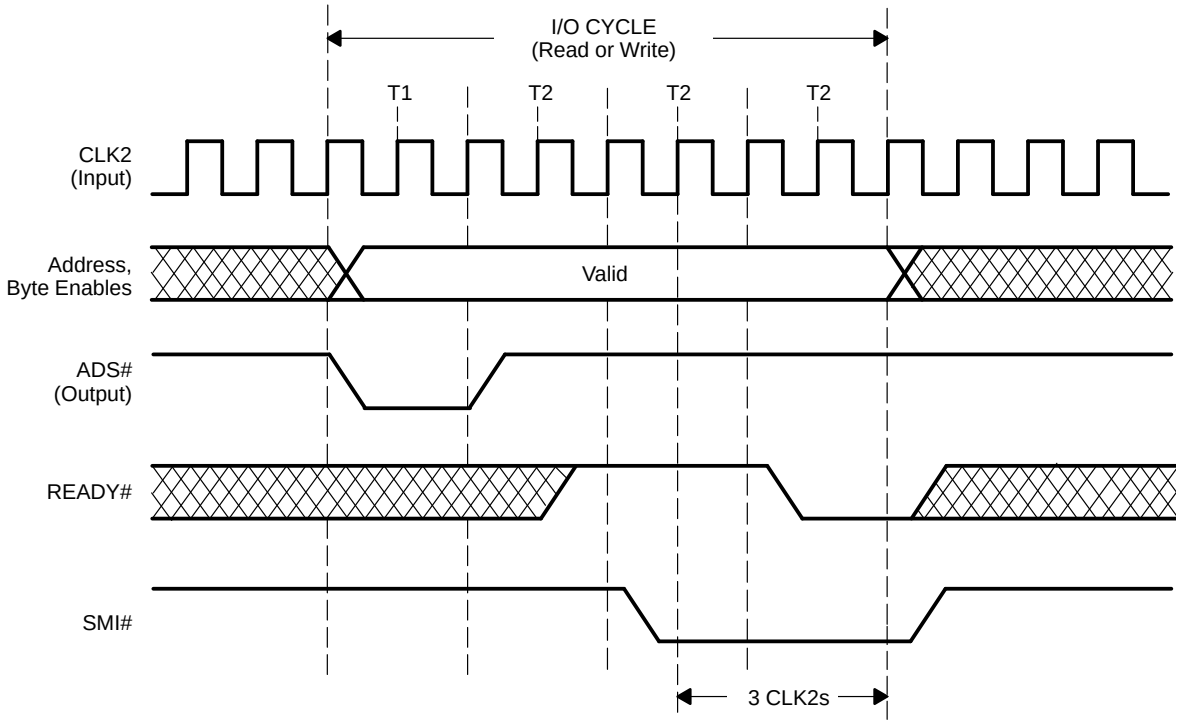


4.4.11.2 I/O Trapping

The TI486SXL series provides I/O trapping 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 three CLK2 edges before 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 4–26. The microprocessor immediately traps to the SMI interrupt handler following execution of the I/O instruction. No other instructions are executed between completing the I/O instruction and entering the SMI service routine. The I/O trap mechanism is not active during coprocessor accesses.

Figure 4–26. TI486SXL I/O Trap Timing



4.4.12 Power Management

The power-management features in the TI486SXL(C) family of microprocessors allow a dramatic reduction in the current required when the microprocessor is in suspend mode (typically less than three percent of the operating current). Suspend mode is entered either by a hardware- or software-initiated action.

Using the hardware to initiate suspend mode involves a two-pin handshake using the SUSP# and SUSPA# signals. Using the software involves initiating the suspend mode through execution of the HALT instruction. Additional power management can be achieved by stopping and restarting the input clock. This technique is available because the TI486SXLC series microprocessors are static devices, meaning that the clock can be stopped and restarted without loss of any internal CPU data. See subsection 4.4.12.3, *Stopping the Input Clock*, on page 4-52

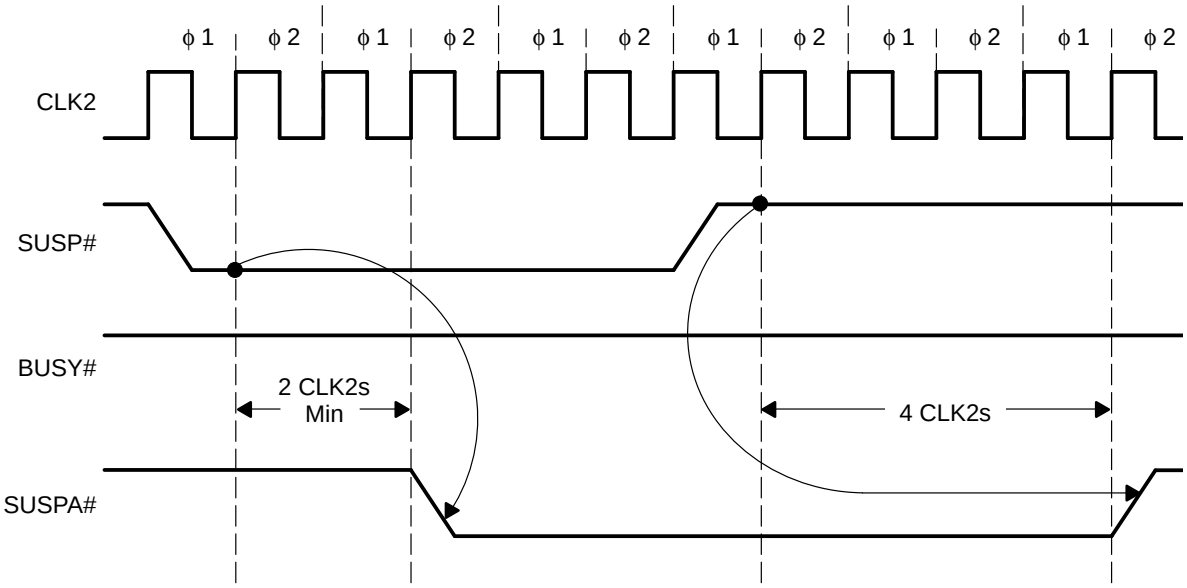
4.4.12.1 SUSP#-Initiated Suspend Mode

The TI486SXL series microprocessor enters suspend mode when the SUSP# input is asserted and execution of the current instruction, any pending de-

coded instructions, and associated bus cycles are completed. The microprocessor also waits for the coprocessor to indicate a not-busy status (BUSY#=1) before entering suspend mode. The SUSPA# output is then asserted. The microprocessor responds to SUSP# and asserts SUSPA# only if the SUSP bit is set in the CCR0 Configuration register.

Figure 4–27 illustrates the microprocessor functional timing for SUSP#-initiated suspend mode. SUSP# is sampled on the phase two ($\phi 2$) CLK2 rising edge and must meet specified setup and hold times to be recognized at a particular CLK2 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 two CLK2 periods. As a maximum, the microprocessor can execute up to two instructions and associated bus cycles before asserting SUSPA#. The time required for the microprocessor to deactivate SUSPA# once SUSP# has been sampled inactive is four CLK2 periods.

Figure 4–27. TI486SXL SUSP#-Initiated Suspend Mode



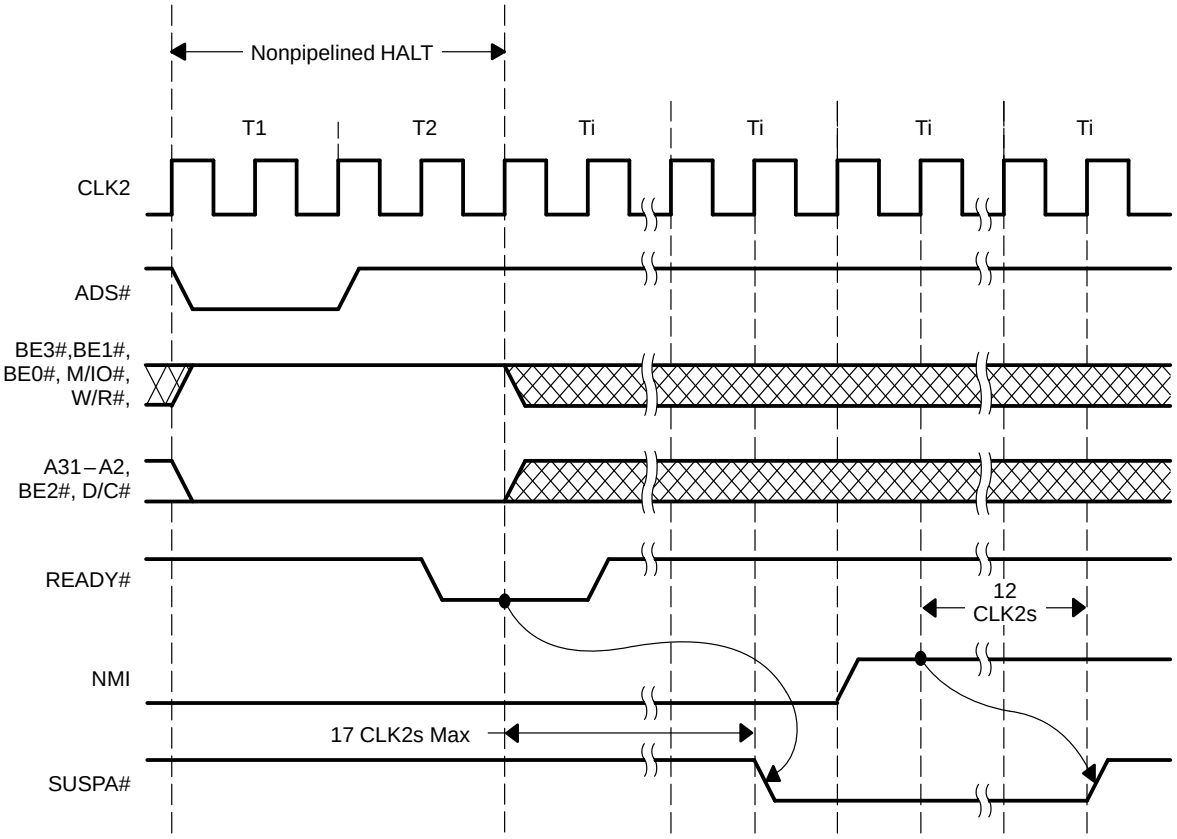
If the microprocessor is in a hold-acknowledge state and SUSP# is asserted, the processor may or may not enter suspend mode depending on the state of the microprocessor internal execution pipeline. If the microprocessor is in a SUSP#-initiated suspend state and the CLK2 input is not stopped, the processor recognizes and acknowledges the HOLD input. The microprocessor stores the occurrence of FLUSH#, NMI, and INTR (if interrupts are enabled) for execution once suspend mode is exited.

4.4.12.2 Halt-Initiated Suspend Mode

The TI486SXL series microprocessor also enters suspend mode as a result of executing a HALT instruction. The SUSPA# output is asserted no more than 17 CLK2 periods following READY# sampled active for the HALT bus cycle as shown in Figure 4–28. Suspend mode is then exited upon recognition of an NMI, SMI#, or an unmasked INTR. SUSPA# is deactivated 12 CLK2 periods

after sampling an active NMI, SMI#, or unmasked INTR. If the microprocessor is in a HALT-initiated suspend mode and the CLK2 input is not stopped, the processor recognizes and acknowledges the HOLD input. The microprocessor stores the occurrence of FLUSH# for execution once suspend mode is exited.

Figure 4–28. TI486SXL HALT-Initiated Suspend Mode



4.4.12.3 Stopping the Input Clock

Because the TI486SXL series microprocessors are static devices, the input clock (CLK2) can be stopped and restarted without loss of any internal CPU data. This assumes that the TI486SXL2 microprocessor is in nonclock-doubled mode when the input clock is stopped. (Refer to subsection 4.2.1, *Clock Doubling Using Software Control*, page 4-15.) CLK2 can be stopped in either phase one ($\phi 1$) or phase two ($\phi 2$) of the clock and in either a logic-high or logic-low state. However, entering suspend mode before stopping CLK2 dramatically reduces the CPU current requirements. Therefore, the recommended sequence for stopping CLK2 in the TI486SXLC2 series microprocessor from clock-doubled mode is:

1) Bring the microprocessor out of clock-doubled mode
2) Initiate suspend mode
3) Wait for the microprocessor to assert SUSPA#
4) Stop the input clock

Note:

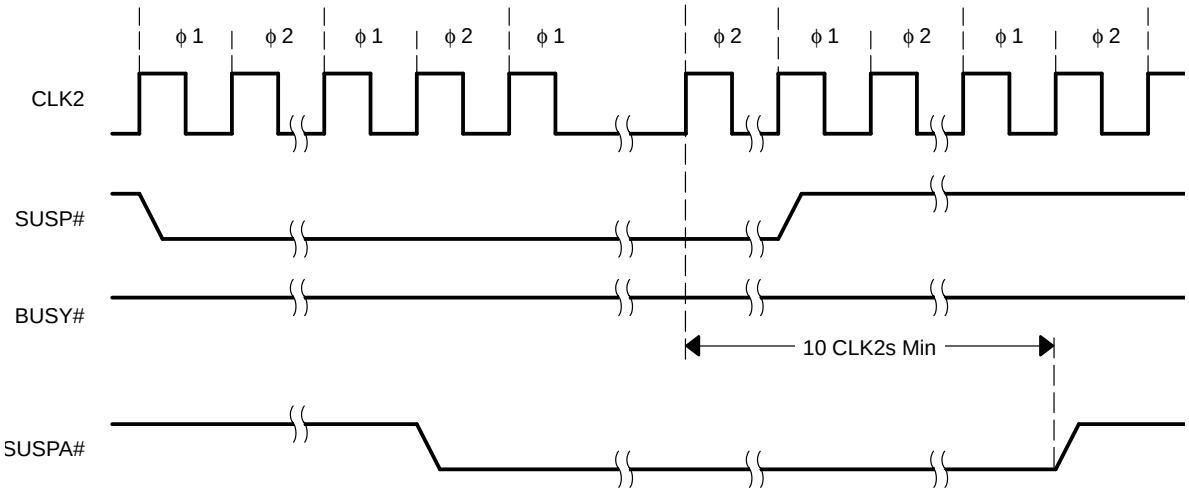
Suspend mode can be entered while in clock-doubled mode as long as CLK2 is not scaled or stopped.

For all other cases, including the TI486SXLC2 in nonclock-doubled mode, the recommended sequence is:

1) Initiate suspend mode
2) Wait for the microprocessor to assert SUSPA#
3) Stop the input clock

The TI486SXL series microprocessor remains suspended until CLK2 is restarted and suspend mode is exited as described above. While CLK2 is stopped, the microprocessor can no longer sample and respond to any input stimulus including the HOLD, FLUSH#, NMI, SMI#, INTR, and RESET inputs. Figure 4–29 illustrates the recommended sequence for stopping CLK2 using SUSP# to initiate suspend mode. CLK2 should be stable for a minimum of 10 clock periods before SUSP# is negated.

Figure 4–29. TI486SXL Stopping CLK2 During Suspend Mode



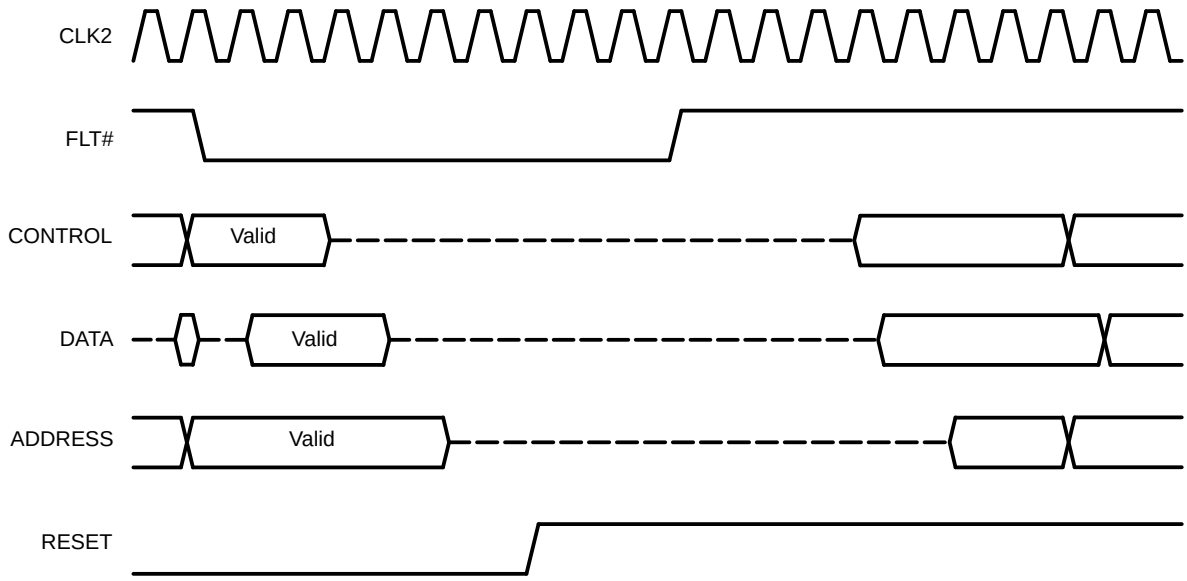
4.4.13 Float (144-Pin QFP and 168-Pin PGA Pinouts Only)

Activating the FLT# input on the 144-pin or 168-pin TI486SXL floats all bidirectional and output signals. Asserting FLT# electrically isolates the microprocessor from the surrounding circuitry. This feature is useful in systems designs that contain an upgrade socket.

FLT# is an asynchronous, active-low input. It is recognized on the rising edge of CLK2. When recognized, it aborts the current bus state and floats the outputs of the microprocessor as shown in Figure 4–30. FLT# must be asserted for a minimum of 16 CLK2 cycles. To exit the float condition, RESET should be asserted and held asserted until after FLT# is negated.

Asserting the FLT# input unconditionally aborts the current bus cycle and forces the microprocessor into the float mode. As a result, the microprocessors are not guaranteed to enter float in a valid state. After deactivating FLT#, the CPU is not guaranteed to exit float in a valid state. The microprocessor RESET input must be asserted before exiting float to ensure that the microprocessor is reset and that it returns in a valid state.

Figure 4–30. TI486SXL Entering and Exiting Float



Chapter 5

Electrical Specifications

This chapter provides electrical specifications for the TI486SXL(C) family of microprocessors. The specifications include electrical connection requirements for all package pins, maximum ratings, recommended operating conditions, dc electrical characteristics, and ac characteristics.

Topic **Page**

5.1 Electrical Connections

This section provides specific requirements for power and ground connections, decoupling, termination of inputs with internal pullup/pulldown resistors, termination of system functional inputs requiring external pullup resistors, termination of unused inputs, and connection to terminals designated NC.

5.1.1 Power and Ground Connections and Decoupling

The high-frequency operation of the TI486SXL(C) microprocessors makes it necessary to install and test the devices using standard high-frequency techniques. The high clock frequencies used in the microprocessors and their 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 making connection to all of the V_{CC}, V_{CC5}, and V_{SS} (GND) terminals.

5.1.2 Pullup/Pulldown Resistors

Table 5–1 lists the input terminals that are internally connected to pullup or pulldown resistors (see Figure 5–1). The pullup resistors are connected to V_{CC}. The pulldown resistors are connected to V_{SS}. Unused inputs do not require connection to external pullup or pulldown resistors.

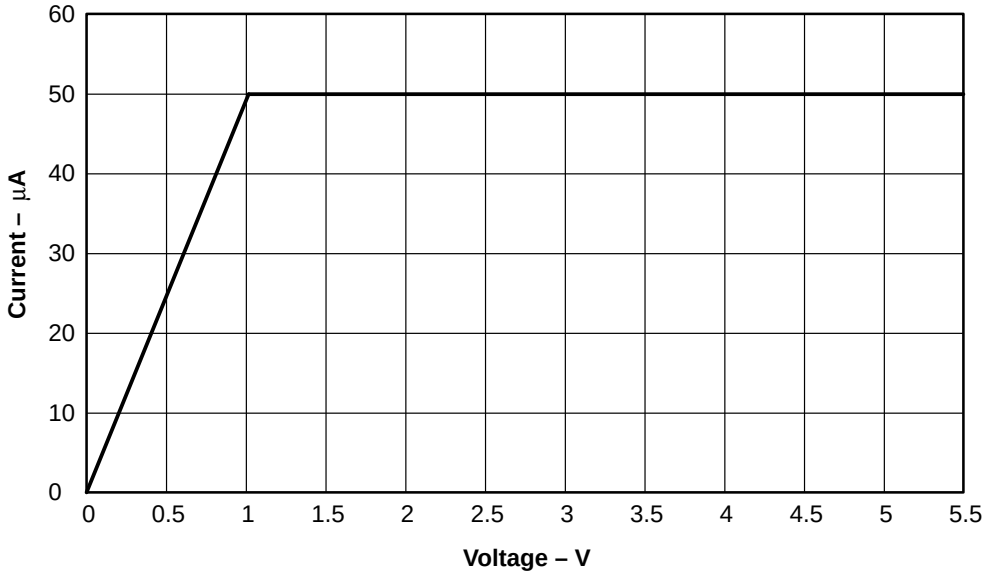
Note:

The internal pullup and pulldown resistors are designed to tie off the individual internal signal associated with that pin. External signals should not be terminated to any of these pins.

Table 5–1. Terminals Connected to Internal Pullup and Pulldown Resistors

Signal	TI486SXLC 100-Terminal	TI486SXL 132-Terminal	TI486SXL 144-Terminal	TI486SXL 168-Terminal	Resistor
A20M#	31	F13	43	D15	Pullup
BUSY#	34	B9	48	S4	Pullup
BS16#	—	C14	115	C17	Pullup
ERROR#	36	A8	49	A12	Pullup
FLT#	28	—	40	C11	Pullup
FLUSH#	30	E13	42	C15	Pullup
KEN#	29	B12	41	F15	Pullup
MEMW#	—	—	66	B16	Pullup
PEREQ	37	C8	50	R17	Pulldown
SMI#	47	C7	67	B10	Pullup
SUSP#	43	A4	63	C13	Pullup

Figure 5–1. Internal Pullup/Pulldown-IV Characteristic



Connect the ADS# and LOCK# output terminals to pullup resistors, as indicated in Table 5–2. The external pullups ensure that the signals remain negated during hold-acknowledge states.

Table 5–2. Terminals Requiring External Pullup Resistors

Signal	TI486SXLC 100-Terminal	TI486SXL 132-Terminal	TI486SXL 144-Terminal	TI486SXL 168-Terminal	External Resistor
ADS#	16	E14	26	S17	20-kΩ pullup
LOCK#	26	C10	38	N15	20-kΩ pullup

5.1.3 NC Designated Terminals

Terminals designated NC should be left disconnected. Connecting or terminating any NC terminal(s) to a pullup resistor, pulldown resistor, or an active signal can cause unpredictable results or nonperformance of the microprocessor.

5.1.4 Unused Signal Input Terminals

All signal inputs not used by the system designer and not listed in Table 5–1 should be connected either to V_{SS} or to V_{CC}. Connect active-high inputs to V_{SS} through a 20-kΩ (±10%) pulldown resistor and active-low inputs to V_{CC} through a 20-kΩ (±10%) pullup resistor to prevent possible spurious operation.

5.2 Absolute Maximum Ratings

The absolute maximum ratings provide specific limits regarding power supply and input voltages, input and output current limits, and operating and storage temperatures.

Table 5–3 specifies the absolute maximum ratings for the TI486SXL(C) family of microprocessors.

Table 5–3. Absolute Maximum Ratings Over Operating Free-Air Temperature Range (Unless Otherwise Noted)†

Parameter			Min	Max	Unit
Supply voltage, V_{CC}	TI486SXLC and TI486SXL	With respect to V_{SS}	–0.5	6.5	V
	TI486SXLC-V, TI486SXL-V, TI486SXLC-G, and TI486SXL-G	With respect to V_{SS}	–0.3	5.5	V
Voltage on any terminal		With respect to V_{SS}	–0.5	V_{CC}+0.5	V
Case temperature		Power applied	–65	110	°C
Storage temperature		No bias	–65	150	°C

† Stresses beyond those listed under absolute maximum ratings may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under recommended operating conditions is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

5.3 Recommended Operating Conditions

Recommended operating conditions provide specific values for power supply and input voltages, required input threshold ranges, output drive currents available for system interfacing, and operating levels for clamp currents and case temperature.

5.3.1 3.3-Volt Microprocessors With 5-Volt Tolerant Inputs, Outputs, and I/Os

Table 5–4 presents the recommended operating conditions for the TI486SXL-G 3.3-V microprocessors with 5-V-tolerant inputs, outputs, and I/Os.

During power up and power down conditions, the 3.3-V V_{CC} terminals and the 5-V V_{CC5} terminal should be ramped simultaneously. The 3.3-V V_{CC} voltage should not exceed the 5-V V_{CC5} voltage by more than 1 V or the device may not initialize correctly. Conversely, the 5-V V_{CC5} can exceed the 3.3-V V_{CC} by up to 2.25 V.

Table 5–4. TI486SXL-G Recommended Operating Conditions

				Min	Max	Unit
V_{CC}	Supply voltage	With respect to V_{SS} See Note 1		3	3.6	V
V_{CC5}	Supply voltage	With respect to V_{SS} See Note 2		3	5.25	V
V_{IH}	High-level input voltage			2	V_{CC5}+0.3	V
V_{IL}	Low-level input voltage			–0.3	0.6	V
V_{ILC}	CLK2 low-level input voltage			–0.3	0.5	V
V_{IHC}	CLK2 high-level input voltage			V_{CC}–0.3	V_{CC5}+0.3	V
I_{OH}	High-level output current	$V_{OH} = V_{OH(min)}$			–2	mA
I_{OL}	Low-level output current	$V_{OL} = V_{OL(max)}$			5	mA
PLL_{LOCK}	Phase-locked loop frequency lock range	With respect to CLK2 frequency		32	50	MHz
T_C	Case temperature	Power applied	TI486SXLC in 100-pin QFP	0	85	°C
			TI486SXL in 132- and 168-pin PGA	0	85	
			TI486SXL in 144-pin QFP	0	85	

Notes: 1) V_{CC} should be no more than 1 V greater than V_{CC5} during power up or the device may not initialize correctly.

2) V_{CC5} should be connected to the 3.3-V supply in a 3.3-V-only system. In mixed systems (3.3/5 V) V_{CC5} should be connected to the 5-V supply.

5.3.2 3.3-Volt Microprocessors

Table 5–5 presents the recommended operating conditions for the TI486SXLC-V and TI486SXL-V 3.3-V microprocessors.

Table 5–5. TI486SXLC-V and TI486SXL-V Recommended Operating Conditions

				Min	Max	Unit
V_{CC}	Supply voltage	With respect to V_{SS}		3	3.6	V
V_{IH}	High-level input voltage			2	$V_{CC}+0.3$	V
V_{IL}	Low-level input voltage			–0.3	0.6	V
V_{ILC}	CLK2 low-level input voltage			–0.3	0.5	V
V_{IHC}	CLK2 high-level input voltage			$V_{CC}-0.3$	$V_{CC}+0.3$	V
I_{OH}	High-level output current	$V_{OH} = V_{OH(min)}$			–2	mA
I_{OL}	Low-level output current	$V_{OL}=V_{OL(max)}$			5	mA
PLL_{LOCK}	Phase-locked loop frequency lock range	With respect to CLK2 frequency		32	50	MHz
T_C	Case temperature	Power applied	TI486SXLC in 100-pin QFP	0	85	°C
			TI486SXL in 132- and 168-pin PGA	0	85	
			TI486SXL in 144-pin QFP	0	85	

5.3.3 5-Volt Microprocessors

Table 5–6 presents the recommended operating conditions for the TI486SXLC and TI486SXL 5-V microprocessors.

Table 5–6. TI486SXLC and TI486SXL Recommended Operating Conditions

				Min	Max	Unit
V_{CC}	Supply voltage	With respect to V_{SS}		4.75	5.25	V
V_{IH}	High-level input voltage			2	$V_{CC}+0.3$	V
V_{IL}	Low-level input voltage			–0.3	0.8	V
V_{ILC}	CLK2 low-level input voltage			–0.3	0.8	V
V_{IHC}	CLK2 high-level input voltage			3.7	$V_{CC}+0.3$	V
I_{OH}	High-level output current	$V_{OH}=V_{OH(min)}$			–1	mA
I_{OL}	Low-level output current	$V_{OL} = V_{OL(max)}$			5	mA
PLL_{LOCK}	Phase-locked loop frequency lock range	With respect to CLK2 frequency		32	50	MHz
T_C	Case temperature	Power applied	TI486SXLC in 100-pin QFP	0	100	°C
			TI486SXL in 132- and 168-pin PGA	0	85	
			TI486SXL in 144-pin QFP	0	100	

5.4 DC Electrical Characteristics

The dc electrical characteristics tables provide specific data regarding the capability of the TI486SXL(C) family microprocessors to interface directly with either CMOS- or TTL-type system functions. Devices are offered for operation in 3.3 and 5-volt mixed, 3.3-volt only, and 5-volt only systems.

5.4.1 3.3-Volt Microprocessors With 5-Volt-Tolerant Inputs

Table 5–7 covers the 3.3-V 40, 20-MHz TI486SXL-G40.

Table 5–8 on page 5-8 covers the 3.3-V 50-MHz TI486SXL2-G50.

Table 5–7. TI486SXL-G40 Electrical Characteristics at Recommended Operating Conditions

Parameter		Test Conditions	See Note 1	TI486SXL-G40 Min	TI486SXL-G40 Typ	TI486SXL-G40 Max	Unit
V_{OL}	Low-level output voltage	I_{OL} = 3 mA				0.4	V
V_{OH}	High-level output voltage	I_{OH} = –1 mA		2.4			V
		I_{OH} = –0.2 mA		V_{CC}–0.4			
I_I	Input current (leakage)	V_{IN} = 0, $V_{IN} \geq V_{CC}$	See Note 2			±15	µA
I_{IH}	High-level input current at PEREQ	V_{IN} = 2.4,	See Note 3			200	µA
I_{IL}	Low-level input current	V_{IL} = 0.45 V,	See Note 4			–400	µA
I_{CC}	Supply current (Active mode)	20 MHz (CLK2 = 40 MHz)			300	400	mA
I_{CCSM}	Supply current (Suspend mode)	20 MHz (CLK2 = 40 MHz)	See Note 5		15		mA
I_{CCSS}	Standby supply current	0 MHz, Suspended/CLK2 stopped,	See Note 5		0.1	1	mA
C_I	Input capacitance	f_c = 1 MHz,	See Note 6			10	pF
C_O	Output or I/O capacitance	f_c = 1 MHz,	See Note 6			12	pF
C_{CLK}	Input capacitance CLK2	f_c = 1 MHz,	See Note 6			20	pF

Notes: 1) Typical values are at V_{CC} = 3.3 V, V_{CC5} = 5 V, and T_A = 25°C.

2) Applicable for all input terminals except those with an internal pullup resistor. See Table 5–1.

3) PEREQ has an internal pulldown resistor.

4) Applicable for all inputs that have an internal pullup resistor. See Table 5–1.

5) All inputs at 0 or V_{CC}. All inputs held static except CLK2 as indicated. All outputs unloaded (static I_{OUT} = 0 mA).

6) Not 100% tested

Table 5–8. TI486SXL2-G50 Electrical Characteristics at Recommended Operating Conditions

Parameter		Test Conditions	See Note 1	TI486SXL2-G50 Min	Typ	Max	Unit
V_{OL}	Low-level output voltage	I_{OL} = 3 mA				0.4	V
V_{OH}	High-level output voltage	I_{OH} = –1 mA		2.4			V
		I_{OH} = –0.2 mA		V_{CC}–0.4			
I_I	Input current (leakage)	V_{IN} = 0, $V_{IN} \geq V_{CC}$	See Note 2			±15	µA
I_{IH}	High-level input current at PEREQ	V_{IN} = 2.4,	See Note 3			200	µA
I_{IL}	Low-level input current	V_{IL} = 0.45 V,	See Note 4			–400	µA
I_{CC}	Supply current (Active mode)	25 MHz (CLK2 = 50 MHz)			365	500	mA
I_{CCSM}	Supply current (Suspend mode)	25 MHz (CLK2 = 50 MHz)	See Note 5		20		mA
I_{CCSS}	Standby supply current	0 MHz, Suspended/CLK2 stopped,	See Note 5		0.1	1	mA
C_I	Input capacitance	f_c = 1 MHz,	See Note 6			10	pF
C_O	Output or I/O capacitance	f_c = 1 MHz,	See Note 6			12	pF
C_{CLK}	Input capacitance CLK2	f_c = 1 MHz,	See Note 6			20	pF

Notes: 1) Typical values are at V_{CC} = 3.3 V, V_{CC5} = 5 V, and T_A = 25°C.

2) Applicable for all input terminals except those with an internal pullup resistor. See Table 5–1.

3) PEREQ has an internal pulldown resistor.

4) Applicable for all inputs that have an internal pullup resistor. See Table 5–1.

5) All inputs at 0 or V_{CC}. All inputs held static except CLK2 as indicated. All outputs unloaded (static I_{OUT} = 0 mA).

6) Not 100% tested

5.4.2 3.3-Volt Microprocessors

Table 5–9 covers the 3.3-V 25-MHz TI486SXLC-V25.

Table 5–10 on page 5-10 covers the 3.3-V 40, 20 MHz TI486SXL-V40.

Table 5–11 on page 5-11 covers the 3.3-V 50, 25 MHz TI486SXL2-V50.

Table 5–9. TI486SXLC-V25 Electrical Characteristics at Recommended Operating Conditions

Parameter		Test Conditions	See Note 1	TI486SXLC-V25 Min	Typ	Max	Unit
V_{OL}	Low-level output voltage	I_{OL} = 3 mA				0.4	V
V_{OH}	High-level output voltage	I_{OH} = –1 mA		2.4			V
		I_{OH} = –0.2 mA		V_{CC}–0.4			
I_I	Input current (leakage)	V_{IN} = 0, $V_{IN} \geq V_{CC}$	See Note 2			±15	µA
I_{IH}	High-level input current at PEREQ	V_{IN} = 2.4,	See Note 3			200	µA
I_{IL}	Low-level input current	V_{IL} = 0.45 V,	See Note 4			–400	µA
I_{CC}	Supply current (Active mode)	25 MHz			225	285	mA
I_{CCSM}	Supply current (Suspend mode)	25 MHz	See Note 5		6		mA
I_{CCSS}	Standby supply current	0 MHz, Suspended/CLK2 stopped,	See Note 5		0.1	1	mA
C_I	Input capacitance	f_c = 1 MHz,	See Note 6			10	pF
C_O	Output or I/O capacitance	f_c = 1 MHz,	See Note 6			12	pF
C_{CLK}	Input capacitance CLK2	f_c = 1 MHz,	See Note 6			20	pF

Notes: 1) Typical values are at V_{CC} = 3.3 V and T_A = 25°C.

2) Applicable for all input terminals except those with an internal pullup resistor. See Table 5–1.

3) PEREQ input has an internal pulldown resistor.

4) Applicable for all inputs that have an internal pullup resistor. See Table 5–1.

5) All inputs at 0 or V_{CC}. All inputs held static except CLK2 as indicated. All outputs unloaded (static I_{OUT} = 0 mA).

6) Not 100% tested

Table 5–10. TI486SXL-V40 Electrical Characteristics at Recommended Operating Conditions

Parameter		Test Conditions	See Note 1	TI486SXL-V40 Min	Typ	Max	Unit
V_{OL}	Low-level output voltage	I_{OL} = 3 mA				0.4	V
V_{OH}	High-level output voltage	I_{OH} = –1 mA		2.4			V
		I_{OH} = –0.2 mA		V_{CC}–0.4			
I_I	Input current (leakage)	V_{IN} = 0, $V_{IN} \geq V_{CC}$	See Note 2			±15	µA
I_{IH}	High-level input current at PEREQ	V_{IN} = 2.4,	See Note 3			200	µA
I_{IL}	Low-level input current	V_{IL} = 0.45 V,	See Note 4			–400	µA
I_{CC}	Supply current (Active mode)	20 MHz (CLK2 = 40 MHz)			300	400	mA
I_{CCSM}	Supply current (Suspend mode)	20 MHz (CLK2 = 40 MHz)	See Note 5		15		mA
I_{CCSS}	Standby supply current	0 MHz, Suspended/CLK2 stopped,	See Note 5		0.1	1	mA
C_I	Input capacitance	f_c = 1 MHz,	See Note 6			10	pF
C_O	Output or I/O capacitance	f_c = 1 MHz,	See Note 6			12	pF
C_{CLK}	Input capacitance CLK2	f_c = 1 MHz,	See Note 6			20	pF

Notes: 1) Typical values are at V_{CC} = 3.3 V and T_A = 25°C.

2) Applicable for all input terminals except those with an internal pullup resistor. See Table 5–1.

3) PEREQ has an internal pulldown resistor.

4) Applicable for all inputs that have an internal pullup resistor. See Table 5–1.

5) All inputs at 0 or V_{CC}. All inputs held static except CLK2 as indicated. All outputs unloaded (static I_{OUT} = 0 mA).

6) Not 100% tested

Table 5–11. TI486SXL2-V50 Electrical Characteristics at Recommended Operating Conditions

Parameter		Test Conditions	See Note 1	TI486SXL2-V50 Min	Typ	Max	Unit
V_{OL}	Low-level output voltage	I_{OL} = 3 mA				0.4	V
V_{OH}	High-level output voltage	I_{OH} = –1 mA		2.4			V
		I_{OH} = –0.2 mA		V_{CC}–0.4			
I_I	Input current (leakage)	V_{IN} = 0, $V_{IN} \geq V_{CC}$	See Note 2			±15	µA
I_{IH}	High-level input current at PEREQ	V_{IN} = 2.4,	See Note 3			200	µA
I_{IL}	Low-level input current	V_{IL} = 0.45 V,	See Note 4			–400	µA
I_{CC}	Supply current (Active mode)	25 MHz (CLK2 = 50 MHz)			365	500	mA
I_{CCSM}	Supply current (Suspend mode)	25 MHz (CLK2 = 50 MHz)	See Note 5		20		mA
I_{CCSS}	Standby supply current	0 MHz, Suspended/CLK2 stopped,	See Note 5		0.1	1	mA
C_I	Input capacitance	f_c = 1 MHz,	See Note 6			10	pF
C_O	Output or I/O capacitance	f_c = 1 MHz,	See Note 6			12	pF
C_{CLK}	Input capacitance CLK2	f_c = 1 MHz,	See Note 6			20	pF

Notes: 1) Typical values are at V_{CC} = 3.3 V and T_A = 25°C.

2) Applicable for all input terminals except those with an internal pullup resistor. See Table 5–1.

3) PEREQ has an internal pulldown resistor.

4) Applicable for all inputs that have an internal pullup resistor. See Table 5–1.

5) All inputs at 0 or V_{CC}. All inputs held static except CLK2 as indicated. All outputs unloaded (static I_{OUT} = 0 mA).

6) Not 100% tested

5.4.3 5-Volt Microprocessors

Table 5–12 covers the 5-V 40, 20-MHz TI486SXLC-040.

Table 5–13 on page 5-13 covers the 5-V 50, 25-MHz TI486SXLC2-050.

Table 5–14 on page 5-14 covers the 5-V 40, 20-MHz TI486SXL-040.

Table 5–15 on page 5-15 covers the 5-V 50. 25-MHz TI486SXL2-050.

Table 5–12. TI486SXLC-040 Electrical Characteristics at Recommended Operating Conditions

Parameter		Test Conditions	See Note 1	TI486SXLC-040 Min	Typ	Max	Unit
V_{OL}	Low-level output voltage	I_{OL} = 5 mA				0.4	V
V_{OH}	High-level output voltage	I_{OH} = –1 mA		2.4			V
		I_{OH} = –0.2 mA		V_{CC}–0.5			
I_I	Input current (leakage)	V_{IN} = 0, $V_{IN} \geq V_{CC}$	See Note 2			±15	µA
I_{IH}	High-level input current at PEREQ	V_{IN} = 2.4,	See Note 3			200	µA
I_{IL}	Low-level input current	V_{IL} = 0.45 V,	See Note 4			–400	µA
I_{CC}	Supply current (Active mode)	20 MHz (CLK2 = 40 MHz)			580	725	mA
I_{CCSM}	Supply current (Suspend mode)	20 MHz (CLK2 = 40 MHz)	See Note 5		10		mA
I_{CCSS}	Standby supply current	0 MHz, Suspended/CLK2 stopped	See Note 5		0.1	1	mA
C_I	Input capacitance	f_c = 1 MHz,	See Note 6			10	pF
C_O	Output or I/O capacitance	f_c = 1 MHz,	See Note 6			12	pF
C_{CLK}	Input capacitance CLK2	f_c = 1 MHz,	See Note 6			20	pF

Notes: 1) Typical values are at V_{CC} = 3.3 V and T_A = 25°C.

2) Applicable for all input terminals except those with an internal pullup resistor. See Table 5–1.

3) PEREQ has an internal pulldown resistor.

4) Applicable for all inputs that have an internal pullup resistor. See Table 5–1.

5) All inputs at 0 or V_{CC}. All inputs held static except CLK2 as indicated. All outputs unloaded (static I_{OUT} = 0 mA).

6) Not 100% tested

Table 5–13. TI486SXLC2-050 Electrical Characteristics at Recommended Operating Conditions

Parameter		Test Conditions	See Note 1	TI486SXLC2-050 Min	Typ	Max	Unit
V_{OL}	Low-level output voltage	I_{OL} = 5 mA				0.45	V
V_{OH}	High-level output voltage	I_{OH} = –1 mA		2.4			V
		I_{OH} = –0.2 mA		V_{CC}–0.5			
I_I	Input current (leakage)	V_{IN} = 0, $V_{IN} \geq V_{CC}$	See Note 2			±15	µA
I_{IH}	High-level input current at PEREQ	V_{IN} = 2.4,	See Note 3			200	µA
I_{IL}	Low-level input current	V_{IL} = 0.45 V,	See Note 4			–400	µA
I_{CC}	Supply current (Active mode)	25 MHz (CLK2 = 50 MHz)			640	850	mA
I_{CCSM}	Supply current (Suspend mode)	25 MHz (CLK2 = 50 MHz)	See Note 5		9		mA
I_{CCSS}	Standby supply current	0 MHz, Suspended/CLK2 stopped,	See Note 5		0.1	1	mA
C_I	Input capacitance	f_c = 1 MHz,	See Note 6			10	pF
C_O	Output or I/O capacitance	f_c = 1 MHz,	See Note 6			12	pF
C_{CLK}	Input capacitance CLK2	f_c = 1 MHz,	See Note 6			20	pF

Notes: 1) Typical values are at V_{CC} = 3.3 V and T_A = 25°C.

2) Applicable for all input terminals except those with an internal pullup resistor. See Table 5–1.

3) PEREQ has an internal pulldown resistor.

4) Applicable for all inputs that have an internal pullup resistor. See Table 5–1.

5) All inputs at 0.4 or V_{CC}–0.4 (CMOS levels). All inputs held static except CLK2 as indicated. All outputs unloaded (static I_{OUT} = 0 mA).

6) Not 100% tested

Table 5–14. TI486SXL-040 Electrical Characteristics at Recommended Operating Conditions

Parameter		Test Conditions	See Note 1	TI486SXL-040 Min	Typ	Max	Unit
V_{OL}	Low-level output voltage	I_{OL} = 5 mA				0.45	V
V_{OH}	High-level output voltage	I_{OH} = –1 mA		2.4			V
		I_{OH} = –0.2 mA		V_{CC}–0.5			
I_I	Input current (leakage)	V_{IN} = 0, $V_{IN} \geq V_{CC}$	See Note 2			±15	µA
I_{IH}	High-level input current at PEREQ	V_{IN} = 2.4,	See Note 3			200	µA
I_{IL}	Low-level input current	V_{IL} = 0.45 V,	See Note 4			–400	µA
I_{CC}	Supply current (Active mode)	20 MHz (CLK2 = 40 MHz)			600	800	mA
I_{CCSM}	Supply current (Suspend mode)	20 MHz (CLK2 = 40 MHz)	See Note 5		10		mA
I_{CCSS}	Standby supply current	0 MHz, Suspended/CLK2 stopped,	See Note 5		0.1	1	mA
C_I	Input capacitance	f_c = 1 MHz,	See Note 6			10	pF
C_O	Output or I/O capacitance	f_c = 1 MHz,	See Note 6			12	pF
C_{CLK}	Input capacitance CLK2	f_c = 1 MHz,	See Note 6			20	pF

Notes: 1) Typical values are at nominal V_{CC} = 3.3 V and T_A = 25°C.

2) Applicable for all input terminals except those with an internal pullup resistor. See Table 5–1.

3) PEREQ input has an internal pulldown resistor.

4) Applicable for all inputs that have an internal pullup resistor. See Table 5–1.

5) All inputs at 0 or V_{CC}. All inputs held static except CLK2 as indicated. All outputs unloaded (static I_{OUT} = 0 mA).

6) Not 100% tested

Table 5–15. TI486SXL2-050 Electrical Characteristics at Recommended Operating Conditions

Parameter		Test Conditions	See Note 1	TI486SXL2-050 Min	Typ	Max	Unit
V_{OL}	Low-level output voltage	I_{OL} = 5 mA				0.45	V
V_{OH}	High-level output voltage	I_{OH} = –1 mA		2.4			V
		I_{OH} = –0.2 mA		V_{CC}–0.5			
I_I	Input current (leakage)	V_{IN} = 0, $V_{IN} \geq V_{CC}$	See Note 2			±15	µA
I_{IH}	High-level input current at PEREQ	V_{IN} = 2.4,	See Note 3			200	µA
I_{IL}	Low-level input current	V_{IL} = 0.45 V,	See Note 4			–400	µA
I_{CC}	Supply current (Active mode)	25 MHz (CLK2 = 50 MHz)			670	900	mA
I_{CCSM}	Supply current (Suspend mode)	25 MHz (CLK2 = 50 MHz)	See Note 5		10		mA
I_{CCSS}	Standby supply current	0 MHz, Suspended/CLK2 stopped,	See Note 5		0.1	1	mA
C_I	Input capacitance	f_c = 1 MHz,	See Note 6			10	pF
C_O	Output or I/O capacitance	f_c = 1 MHz,	See Note 6			12	pF
C_{CLK}	Input capacitance CLK2	f_c = 1 MHz,	See Note 6			20	pF

Notes: 1) Typical values are at nominal V_{CC} = 3.3 V and T_A = 25°C.

2) Applicable for all input terminals except those with an internal pullup resistor. See Table 5–1.

3) PEREQ input has an internal pulldown resistor.

4) Applicable for all inputs that have an internal pullup resistor. See Table 5–1.

5) All inputs at 0 or V_{CC}. All inputs held static except CLK2 as indicated. All outputs unloaded (static I_{OUT} = 0 mA).

6) Not 100% tested

5.5 AC Characteristics

The ac characteristics provide detailed information regarding measurement points, specific requirements for setup and hold times, and propagation delay times of the TI486SXL(C) microprocessors.

5.5.1 Measurement Points for AC Characteristics

The rising-clock-edge reference level V_{refC}, and other reference levels are specified in Table 5–16 for the TI486SXL(C) family of microprocessors. Input or output signals must cross these levels during testing.

Table 5–16. Measurement Points for AC Characteristics

Symbol	TI486SXLC-V and TI486SXL-V	TI486SXLC and TI486SXL	Unit
V_{refC}	1.5	2	V
V_{ref}	1.2	1.5	V
V_{IHC}	$V_{CC}-0.3$	$V_{CC}-0.8$	V
V_{ILC}	0.6	0.8	V
V_{IHD}	2.3	3	V
V_{ILD}	0	0	V

Figure 5–2 and Figure 5–3 show delays (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 during which a synchronous input signal must be stable for correct operation.

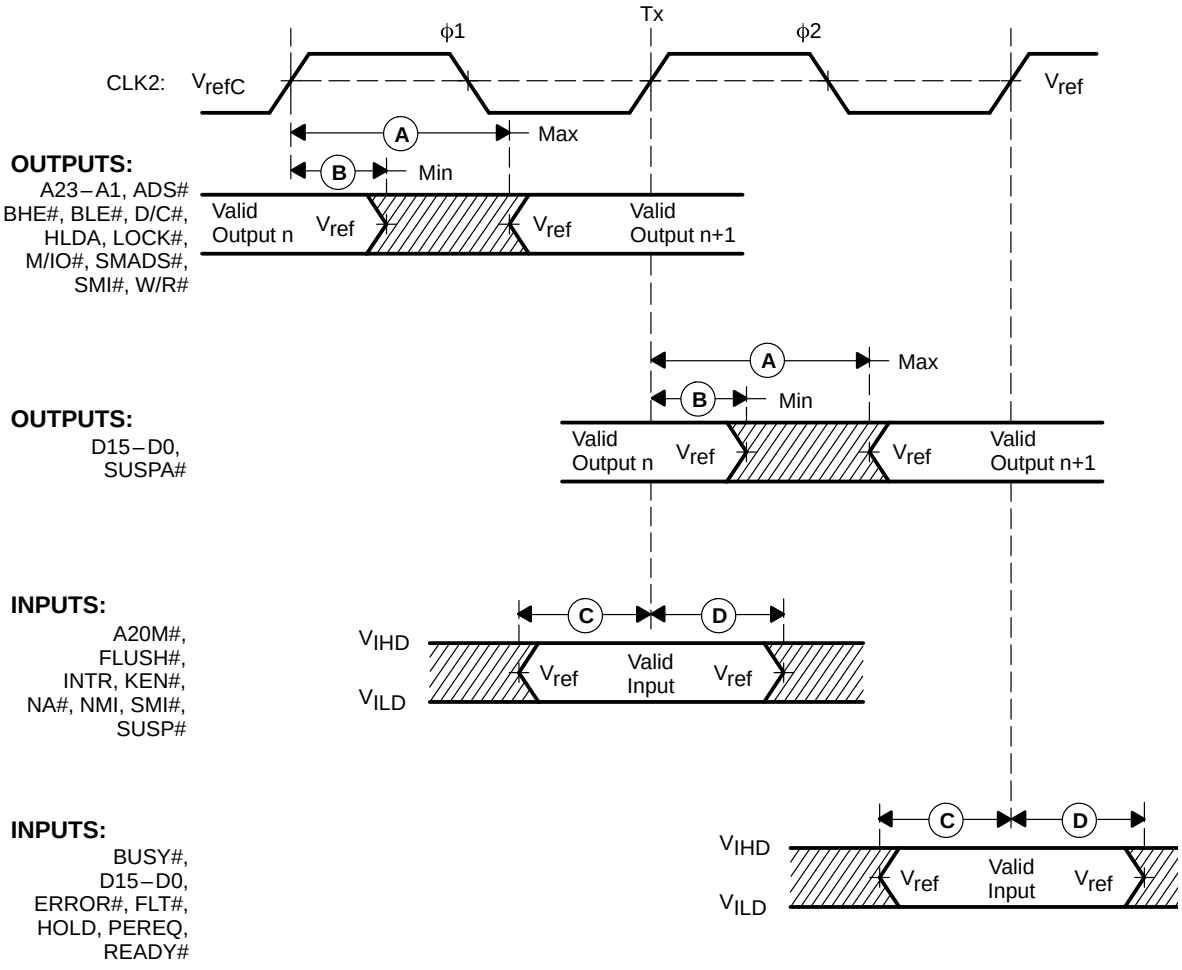
The TI486SXLC microprocessor outputs A23–A1, ADS#, BHE#, BLE#, D/C#, HLDA, LOCK#, M/IO#, SMADS#, SMI#, and W/R# change only at the beginning of phase one (Figure 5–2, $\phi 1$). Outputs D15–D0 (write cycles) and SUSPA# change at the beginning of phase two ($\phi 2$).

The TI486SXLC microprocessor inputs BUSY#, D15–D0 (read cycles), ERROR#, FLT#, HOLD, PEREQ, and READY# are sampled at the beginning of phase one (Figure 5–2, $\phi 1$). Inputs A20M#, FLUSH#, INTR, KEN#, NA#, NMI, SMI# and SUSP# are sampled at the beginning of phase two ($\phi 2$).

The TI486SXL microprocessor outputs A31–A2, ADS#, BE3#–BE0#, D/C#, HLDA, LOCK#, M/IO#, SMADS#, SMI#, and W/R# change only at the beginning of phase one (Figure 5–3, $\phi 1$). Outputs D31–D0 (write cycles) and SUSPA# change at the beginning of phase two ($\phi 2$).

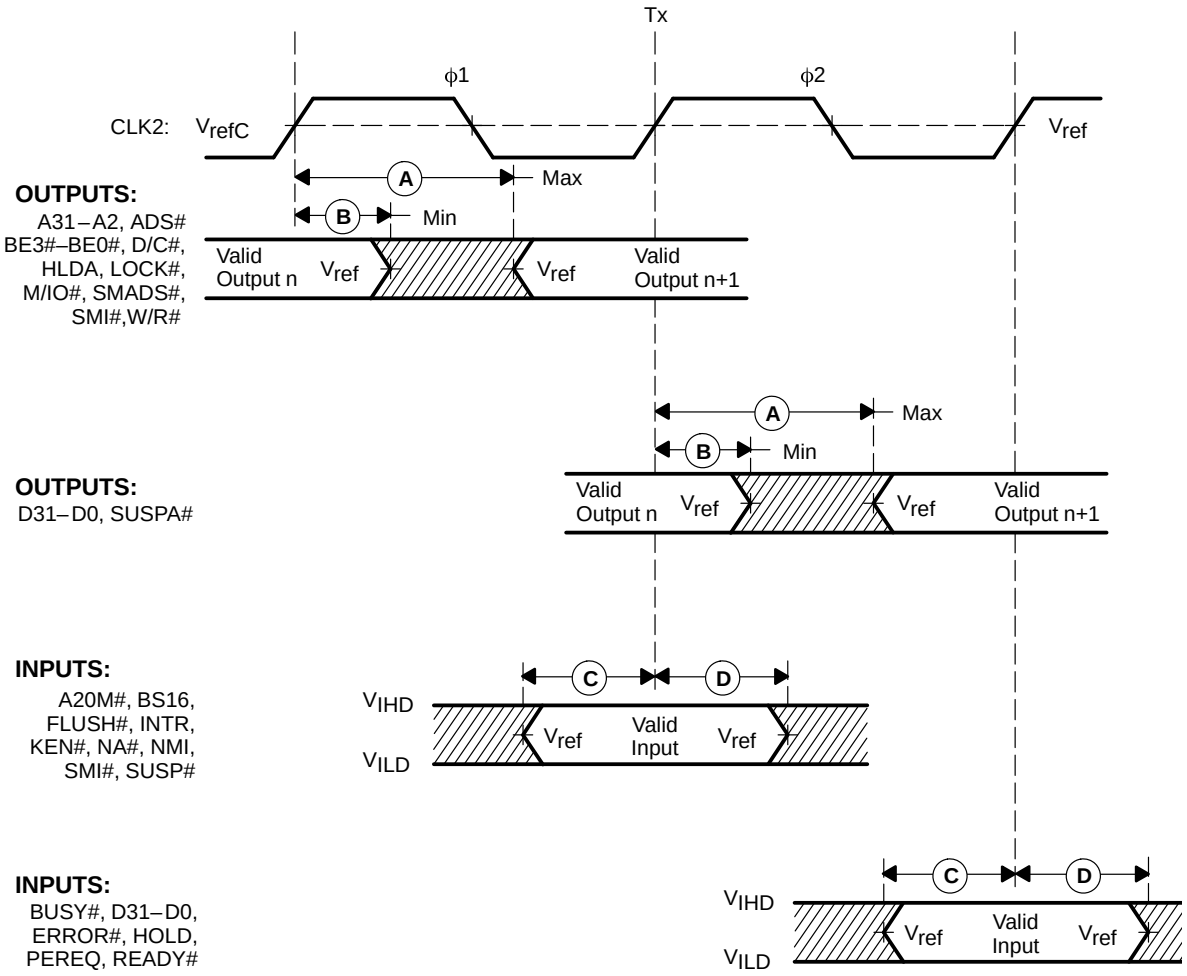
The TI486SXL microprocessor inputs BUSY#, D31–D0 (read cycles), ERROR#, HOLD, PEREQ, and READY# are sampled at the beginning of phase 1 (Figure 5–3, $\phi 1$). Inputs A20M#, BS16, FLUSH#, INTR, KEN#, NA#, NMI, SMI# and SUSP# are sampled at the beginning of phase two ($\phi 2$).

Figure 5–2. TI486SXLC Drive Level and Measurement Points for AC Characteristics



Legend: A – Maximum Output Delay Specification
B – Minimum Output Delay Specification
C – Minimum Input Setup Specification
D – Minimum Input Hold Specificaton

Figure 5–3. TI486SXL Drive Level and Measurement Points for AC Characteristics

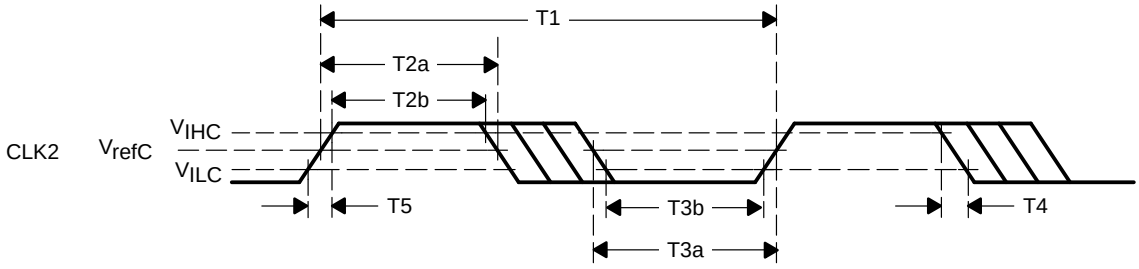


Legend: A – Maximum Output Delay Specification
B – Minimum Output Delay Specification
C – Minimum Input Setup Specification
D – Minimum Input Hold Specificaton

5.5.2 CLK2 Timing Measurement Points

The CLK2 timing measurement points are illustrated in Figure 5–4 for the TI486SXL(C) family of microprocessors.

Figure 5–4. CLK2 Timing Measurement Points



5.5.3 AC Data Characteristics Tables

Parametric ac characteristics include output delays, input setup requirements, input hold requirements, and output float delays. These characteristics are based on the measurement points identified in Figure 5–2 on page 5-17, Figure 5–3 on page 5-18, and Figure 5–4.

5.5.3.1 AC Data for 3.3-Volt Microprocessors With 5-Volt Tolerant Inputs

Table 5–17 covers the 3.3-V 40, 20-MHz TI486SXL-G40.

Table 5–18 on page 5-21 covers the 3.3-V 50-MHz TI486SXL2-G50.

Table 5–17. AC Characteristics for TI486SXL-G40 (See Note 1)

Symbol	Parameter	TI486SXLG40		Unit	Figure	Notes
		MIN	MAX			
	CLK2 frequency range	32	40	MHz		
T1	CLK2 period	12.5		ns	5-4	Note 2
T2a	CLK2 high time	5			5-4	Note 3
T2b	CLK2 high time	3.25			5-4	Note 3
T3a	CLK2 low time	5			5-4	Note 3
T3b	CLK2 low time	3.25			5-4	Note 3
T4	CLK2 fall time		4		5-4	Note 3
T5	CLK2 rise time		4		5-4	Note 3
T6	A31–A2 valid delay	3	12.5	ns	5-12, 5-15	C_L = 50 pF
T6a	SMI# valid delay	3	12.5		5-12, 5-15	C_L = 50 pF
T7	A31–A2 float delay	3	17		5-15	Note 4
T8	BE3# – BE0#, LOCK# valid delay	3	12.5	ns	5-12, 5-15	C_L = 50 pF
T9	BE3# – BE0#, LOCK# float delay	3	17		5-15	Note 4
T10	ADS#, D/C#, M/IO#, W/R# valid delay	3	12.5	ns	5-12, 5-15	C_L = 50 pF
T10a	SMADS# valid delay	3	12.5		5-12, 5-15	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, W/R# float delay	3	17	ns	5-15	Note 4
T11a	SMADS# float delay	3	17		5-15	Note 4
T12	D31–D0 write data, SUSPA# valid delay	5	20	ns	5-12, 5-13	C_L = 50 pF,
T12a	D31–D0 write data hold time	2			5-14	Note 5
T13	D31–D0 write data, SUSPA# float delay	3	14.5		5-15	Notes 4, 6
T14	HDLA valid delay	3	17	ns	5-15	C_L = 50 pF
T15	A20M#, FLUSH#, KEN#, NA#, SUSP# setup time	5		ns	5-11	
T16	A20M#, FLUSH#, KEN#, NA#, SUSP# hold time	2			5-11	
T17	BS16# setup time	5		ns	5-11	
T18	BS16# hold time	2			5-11	
T19	READY# setup time	5		ns	5-11	
T20	READY# hold time	3			5-11	
T21	D31–D0 read data setup time	5		ns	5-11	
T22	D31–D0 read data hold time	3			5-11	
T23	HOLD setup time	4		ns	5-11	
T24	HOLD hold time	2			5-11	
T25	RESET setup time	4.5		ns	5-5	
T26	RESET hold time	2			5-5	Note 5
T27	NMI, INTR setup time	5		ns	5-10	Note 7
T27a	SMI# setup time	5			5-10	Note 7
T28	NMI, INTR hold time	5			5-10	Note 7
T28a	SMI# hold time	5			5-10	Note 7
T29	PEREQ, ERROR#, BUSY# setup time	5		ns	5-10	Note 7
T30	PEREQ, ERROR#, BUSY# hold time	3			5-10	Note 7
T31	Clock-doubled PLL lock time		20	μs		Note 8

Notes:
1) V_{CC} = 3 V to 3.6 V, V_{CC5} = 4.75 V to 5.25 V or 3 V to 3.6 V, T_C = 0 to 85 °C
2) Input clock can be stopped; therefore, minimum CLK2 frequency is 0 MHz.
3) These parameters are not tested. They are determined by design characterization.
4) Float condition occurs when maximum output current becomes less than I_I in magnitude. Float is not 100% tested.
5) Not 100% tested.
6) SUSPA# floats only in response to activation of FLT#. SUSPA# does not float during a hold-acknowledge state.
7) These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes to assure recognition within a specific CLK2 period.
8) Delay time from setting CKD in CCR0 to entering clock-doubled mode.

Table 5–18. AC Characteristics for TI486SXL2-G50 (See Note 1)

Symbol	Parameter	TI486SXL2-G50 MIN	TI486SXL2-G50 MAX	Unit	Figure	Notes
	CLK2 clock-doubled frequency range	32	50	MHz		
T1	CLK2 period	20		ns	5-4	Note 2
T2a	CLK2 high time	7			5-4	Note 3
T2b	CLK2 high time	4			5-4	Note 3
T3a	CLK2 low time	7			5-4	Note 3
T3b	CLK2 low time	5			5-4	Note 3
T4	CLK2 fall time		7		5-4	Note 3
T5	CLK2 rise time		7		5-4	Note 3
T6	A31–A2 valid delay	3	21	ns	5-12, 5-15	C_L = 50 pF
T6a	SMI# valid delay	3	30		5-12, 5-15	C_L = 50 pF
T7	A31–A2 float delay	4	30		5-15	Note 4
T8	BE3# – BE0#, LOCK# valid delay	2.5	18	ns	5-12, 5-15	C_L = 50 pF
T9	BE3# – BE0#, LOCK# float delay	4	30		5-15	Note 4
T10	ADS#, D/C#, M/IO#, W/R# valid delay	4	19	ns	5-12, 5-15	C_L = 50 pF
T10a	SMADS# valid delay	4	19		5-12, 5-15	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, W/R# float delay	4	30	ns	5-15	Note 4
T11a	SMADS# float delay	4	30		5-15	Note 4
T12	D31–D0 write data, SUSPA# valid delay	3.5	27	ns	5-12, 5-13	C_L = 50 pF,
T12a	D31–D0 write data hold time	2			5-14	Note 5
T13	D31–D0 write data, SUSPA# float delay	4	22		5-15	Notes 4, 6
T14	HDLA valid delay	2	22	ns	5-15	C_L = 50 pF
T15	A20M#, FLUSH#, KEN#, NA#, SUSP# setup time	5		ns	5-11	
T16	A20M#, FLUSH#, KEN#, NA#, SUSP# hold time	3.5			5-11	
T17	BS16# setup time	7		ns	5-11	
T18	BS16# hold time	2			5-11	
T19	READY# setup time	9		ns	5-11	
T20	READY# hold time	4			5-11	
T21	D31–D0 read data setup time	7		ns	5-11	
T22	D31–D0 read data hold time	5			5-11	
T23	HOLD setup time	9		ns	5-11	
T24	HOLD hold time	3.5			5-11	
T25	RESET setup time	8		ns	5-5	
T26	RESET hold time	3			5-5	Note 5
T27	NMI, INTR setup time	6		ns	5-10	Note 7
T27a	SMI# setup time	6			5-10	Note 7
T28	NMI, INTR hold time	6			5-10	Note 7
T28a	SMI# hold time	6			5-10	Note 7
T29	PEREQ, ERROR#, BUSY# setup time	6		ns	5-10	Note 7
T30	PEREQ, ERROR#, BUSY# hold time	5			5-10	Note 7
T31	Clock-doubled PLL lock time		20	µs		Note 8

Notes:
1) V_{CC} = 3 V to 3.6 V, V_{CC5} = 4.75 V to 5.25 V or 3 V to 3.6 V, T_C = 0 to 85 °C
2) Input clock can be stopped; therefore, minimum CLK2 frequency is 0 MHz.
3) These parameters are not tested. They are determined by design characterization.
4) Float condition occurs when maximum output current becomes less than I_I in magnitude. Float is not 100% tested.
5) Not 100% tested.
6) SUSPA# floats only in response to activation of FLT#. SUSPA# does not float during a hold-acknowledge state.
7) These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes to assure recognition within a specific CLK2 period.
8) Delay time from setting CKD in CCR0 to entering clock-doubled mode.

5.5.3.2 AC Data for 3.3-Volt Microprocessors

Table 5–19 covers the 3.3-V 25-MHz TI486SXLC-V25.

Table 5–20 on page 5-23 covers the 3.3-V 40, 20 MHz TI486SXL-V40.

Table 5–21 on page 5-24 covers the 3.3-V 50 MHZ TI486SXL2-050.

Table 5–19. AC Characteristics for TI486SXLC-V25 (See Note 1)

Symbol	Parameter	TI486SXLC-V25 MIN	TI486SXLC-V25 MAX	Unit	Figure	Notes
T1	CLK2 period	20			5-4	Note 2
T2a	CLK2 high time	7			5-4	Note 3
T2b	CLK2 high time	4			5-4	Note 3
T3a	CLK2 low time	7		ns	5-4	Note 3
T3b	CLK2 low time	5			5-4	Note 3
T4	CLK2 fall time		7		5-4	Note 3
T5	CLK2 rise time		7		5-4	Note 3
T6	A23–A1 valid delay	3	21		5-7, 5-10	C_L = 50 pF
T6a	SMI# valid delay	3	30	ns	5-7, 5-10	C_L = 50 pF
T7	A23–A1 float delay	4	30		5-10	Note 4
T8	BHE#, BLE#, LOCK# valid delay	2.5	18	ns	5-7, 5-10	C_L = 50 pF
T9	BHE#, BLE#, LOCK# float delay	4	30		5-10	Note 4
T10	ADS#, D/C#, M/IO#, W/R# valid delay	4	19	ns	5-7, 5-10	C_L = 50 pF
T10a	SMADS# valid delay	4	19		5-7,5-10	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, W/R# float delay	4	30	ns	5-10	Note 4
T11a	SMADS# float delay	4	30		5-10	Note 4
T12	D15–D0 write data, SUSPA# valid delay	3.5	27		5-7, 5-8	C_L = 50 pF,
T12a	D15–D0 write data hold time	2		ns	5-9	Note 5
T13	D15–D0 write data, SUSPA# float delay	4	22		5-10	Notes 4, 6
T14	HDLA valid delay	2	22	ns	5-10	C_L = 50 pF
T15	NA#, SUSP#, FLUSH#, KEN#, A20M# setup time	5		ns	5-6	
T16	NA#, SUSP#, FLUSH#, KEN#, A20M# hold time	3.5			5-6	
T19	READY# setup time	9		ns	5-6	
T20	READY# hold time	4			5-6	
T21	D15–D0 read data setup time	7		ns	5-6	
T22	D15–D0 read data hold time	5			5-6	
T23	HOLD setup time	9		ns	5-6	
T24	HOLD hold time	3.5			5-6	
T25	RESET setup time	8		ns	5-5	
T26	RESET hold time	3			5-5	Note 5
T27	NMI, INTR setup time	6		ns	5-6	Note 7
T27a	SMI# setup time	6			5-6	Note 7
T28	NMI, INTR hold time	6			5-6	Note 7
T28a	SMI# hold time	6			5-6	Note 7
T29	PEREQ, ERROR#, BUSY# setup time	6		ns	5-6	Note 7
T30	PEREQ, ERROR#, BUSY# hold time	5			5-6	Note 7

Notes: 1) V_{CC} = 3 V to 3.6 V, T_C = 0 °C to 85 °C
2) Input clock can be stopped; therefore, minimum CLK2 frequency is 0 MHz.
3) These parameters are not tested. They are determined by design characterization.
4) Float condition occurs when maximum output current becomes less than I_I in magnitude. Float is not 100% tested.
5) Not 100% tested.
6) SUSPA# floats only in response to activation of FLT#. SUSPA# does not float during a hold acknowledge state.
7) These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes to assure recognition within a specific CLK2 period.

Table 5–20. AC Characteristics for TI486SXL-V40 (See Note 1)

Symbol	Parameter	TI486SXL-V40 MIN	TI486SXL-V40 MAX	Unit	Figure	Notes
	CLK2 frequency range	32	40	MHz		
T1	CLK2 period	12.5		ns	5-4	Note 2
T2a	CLK2 high time	5			5-4	Note 3
T2b	CLK2 high time	3.25			5-4	Note 3
T3a	CLK2 low time	5			5-4	Note 3
T3b	CLK2 low time	3.25			5-4	Note 3
T4	CLK2 fall time		4		5-4	Note 3
T5	CLK2 rise time		4		5-4	Note 3
T6	A31–A2 valid delay	3	12.5	ns	5-12, 5-15	C_L = 50 pF
T6a	SMI# valid delay	3	12.5		5-12, 5-15	C_L = 50 pF
T7	A31–A2 float delay	3	17		5-15	Note 4
T8	BE3# – BE0#, LOCK# valid delay	3	12.5	ns	5-12, 5-15	C_L = 50 pF
T9	BE3# – BE0#, LOCK# float delay	3	17		5-15	Note 4
T10	ADS#, D/C#, M/IO#, W/R# valid delay	3	12.5	ns	5-12, 5-15	C_L = 50 pF
T10a	SMADS# valid delay	3	12.5		5-12, 5-15	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, W/R# float delay	3	17	ns	5-15	Note 4
T11a	SMADS# float delay	3	17		5-15	Note 4
T12	D31–D0 write data, SUSPA# valid delay	5	20	ns	5-12, 5-13	C_L = 50 pF,
T12a	D31–D0 write data hold time	2			5-14	Note 5
T13	D31–D0 write data, SUSPA# float delay	3	14.5		5-15	Notes 4, 6
T14	HDLA valid delay	3	17	ns	5-15	C_L = 50 pF
T15	A20M#, FLUSH#, KEN#, NA#, SUSP# setup time	5		ns	5-11	
T16	A20M#, FLUSH#, KEN#, NA#, SUSP# hold time	2			5-11	
T17	BS16# setup time	5		ns	5-11	
T18	BS16# hold time	2			5-11	
T19	READY# setup time	5		ns	5-11	
T20	READY# hold time	3			5-11	
T21	D31–D0 read data setup time	5		ns	5-11	
T22	D31–D0 read data hold time	3			5-11	
T23	HOLD setup time	4		ns	5-11	
T24	HOLD hold time	2			5-11	
T25	RESET setup time	4.5		ns	5-5	
T26	RESET hold time	2			5-5	Note 5
T27	NMI, INTR setup time	5		ns	5-10	Note 7
T27a	SMI# setup time	5			5-10	Note 7
T28	NMI, INTR hold time	5			5-10	Note 7
T28a	SMI# hold time	5			5-10	Note 7
T29	PEREQ, ERROR#, BUSY# setup time	5		ns	5-10	Note 7
T30	PEREQ, ERROR#, BUSY# hold time	3			5-10	Note 7
T31	Clock-doubled PLL lock time		20	µs		Note 8

Notes: 1) V_{CC} = 3 V to 3.6 V, T_C = 0 to 85 °C

Notes: 2) Input clock can be stopped; therefore, minimum CLK2 frequency is 0 MHz.
3) These parameters are not tested. They are determined by design characterization.
4) Float condition occurs when maximum output current becomes less than I_I in magnitude. Float is not 100% tested.
5) Not 100% tested.
6) SUSPA# floats only in response to activation of FLT#. SUSPA# does not float during a hold-acknowledge state.
7) These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes to assure recognition within a specific CLK2 period.
8) Delay time from setting CKD in CCR0 to entering clock-doubled mode.

Table 5–21. AC Characteristics for TI486SXL2-V50 (See Note 1)

Symbol	Parameter	TI486SXL2-V50 MIN	TI486SXL2-V50 MAX	Unit	Figure	Notes
	CLK2 clock-doubled frequency range	32	50	MHz		
T1	CLK2 period	20		ns	5-4	Note 2
T2a	CLK2 high time	7		ns	5-4	Note 3
T2b	CLK2 high time	4		ns	5-4	Note 3
T3a	CLK2 low time	7		ns	5-4	Note 3
T3b	CLK2 low time	5		ns	5-4	Note 3
T4	CLK2 fall time		7	ns	5-4	Note 3
T5	CLK2 rise time		7	ns	5-4	Note 3
T6	A31–A2 valid delay	3	21	ns	5-12, 5-15	C_L = 50 pF
T6a	SMI# valid delay	3	30	ns	5-12, 5-15	C_L = 50 pF
T7	A31–A2 float delay	4	30	ns	5-15	Note 4
T8	BE3# – BE0#, LOCK# valid delay	2.5	18	ns	5-12, 5-15	C_L = 50 pF
T9	BE3# – BE0#, LOCK# float delay	4	30	ns	5-15	Note 4
T10	ADS#, D/C#, M/IO#, W/R# valid delay	4	19	ns	5-12, 5-15	C_L = 50 pF
T10a	SMADS# valid delay	4	19	ns	5-12, 5-15	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, W/R# float delay	4	30	ns	5-15	Note 4
T11a	SMADS# float delay	4	30	ns	5-15	Note 4
T12	D31–D0 write data, SUSPA# valid delay	3.5	27	ns	5-12, 5-13	C_L = 50 pF,
T12a	D31–D0 write data hold time	2		ns	5-14	Note 5
T13	D31–D0 write data, SUSPA# float delay	4	22	ns	5-15	Notes 4, 6
T14	HDLA valid delay	2	22	ns	5-15	C_L = 50 pF
T15	A20M#, FLUSH#, KEN#, NA#, SUSP# setup time	5		ns	5-11	
T16	A20M#, FLUSH#, KEN#, NA#, SUSP# hold time	3.5		ns	5-11	
T17	BS16# setup time	7		ns	5-11	
T18	BS16# hold time	2		ns	5-11	
T19	READY# setup time	9		ns	5-11	
T20	READY# hold time	4		ns	5-11	
T21	D31–D0 read data setup time	7		ns	5-11	
T22	D31–D0 read data hold time	5		ns	5-11	
T23	HOLD setup time	9		ns	5-11	
T24	HOLD hold time	3.5		ns	5-11	
T25	RESET setup time	8		ns	5-5	
T26	RESET hold time	3		ns	5-5	Note 5
T27	NMI, INTR setup time	6		ns	5-10	Note 7
T27a	SMI# setup time	6		ns	5-10	Note 7
T28	NMI, INTR hold time	6		ns	5-10	Note 7
T28a	SMI# hold time	6		ns	5-10	Note 7
T29	PEREQ, ERROR#, BUSY# setup time	6		ns	5-10	Note 7
T30	PEREQ, ERROR#, BUSY# hold time	5		ns	5-10	Note 7
T31	Clock-doubled PLL lock time		20	µs		Note 8

Notes:
1) V_{CC} = 3 V to 3.6 V, T_C = 0 to 85°C
2) Input clock can be stopped; therefore, minimum CLK2 frequency is 0 MHz.
3) These parameters are not tested. They are determined by design characterization.
4) Float condition occurs when maximum output current becomes less than I_I in magnitude. Float is not 100% tested.
5) Not 100% tested.
6) SUSPA# floats only in response to activation of FLT#. SUSPA# does not float during a hold-acknowledge state.
7) These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes to assure recognition within a specific CLK2 period.
8) Delay time from setting CKD in CCR0 to entering clock-doubled mode.

5.5.3.3 AC Data for 5-Volt Microprocessors

Table 5–22 covers the 5-V 40, 20 MHz TI486SXLC-040.

Table 5–23 on page 5-26 covers the 5-V 50 MHz TI486SXLC2-050.

Table 5–24 on page 5-27 covers the 5-V 40, 20 MHz TI486SXL-040.

Table 5–25 on page 5-28 covers the 5-V 50 MHz TI486SXL2-050

Table 5–22. AC Characteristics for TI486SXLC-040 (See Note 1)

Symbol	Parameter	TI486SXLC-040 MIN	TI486SXLC-040 MAX	Unit	Figure	Notes
	CLK2 frequency range	32	40	MHz		
T1	CLK2 period	12.5		ns	5-4	Note 2
T2a	CLK2 high time	5			5-4	Note 3
T2b	CLK2 high time	3.25			5-4	Note 3
T3a	CLK2 low time	5			5-4	Note 3
T3b	CLK2 low time	3.25			5-4	Note 3
T4	CLK2 fall time		4		5-4	Note 3
T5	CLK2 rise time		4		5-4	Note 3
T6	A23–A1 valid delay	3	12.5	ns	5-7, 5-10	C_L = 50 pF
T6a	SMI# valid delay	3	12.5		5-7, 5-10	C_L = 50 pF
T7	A23–A1 float delay	3	17		5-10	Note 4
T8	BHE#, BLE#, LOCK# valid delay	3	12.5	ns	5-7, 5-10	C_L = 50 pF
T9	BHE#, BLE#, LOCK# float delay	3	17		5-10	Note 4
T10	ADS#, D/C#, M/IO#, W/R# valid delay	3	12.5	ns	5-7, 5-10	C_L = 50 pF
T10a	SMADS# valid delay	3	12.5		5-7, 5-10	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, W/R# float delay	3	17	ns	5-10	Note 4
T11a	SMADS# float delay	3	17		5-10	Note 4
T12	D15–D0 write data, SUSPA# valid delay	5	20	ns	5-7, 5-8	C_L = 50 pF,
T12a	D15–D0 write data hold time	2			5-9	Note 5
T13	D15–D0 write data, SUSPA# float delay	3	14.5		5-10	Notes 4, 6
T14	HDLA valid delay	3	17	ns	5-10	C_L = 50 pF
T15	NA#, SUSP#, FLUSH#, KEN#, A20M# setup time	5		ns	5-6	
T16	NA#, SUSP#, FLUSH#, KEN#, A20M# hold time	2			5-6	
T19	READY# setup time	5		ns	5-6	
T20	READY# hold time	3			5-6	
T21	D15–D0 read data setup time	5		ns	5-6	
T22	D15–D0 read data hold time	3			5-6	
T23	HOLD setup time	4		ns	5-6	
T24	HOLD hold time	2			5-6	
T25	RESET setup time	4.5		ns	5-5	
T26	RESET hold time	2			5-5	Note 5
T27	NMI, INTR setup time	5		ns	5-6	Note 7
T27a	SMI# setup time	5			5-6	Note 7
T28	NMI, INTR hold time	5			5-6	Note 7
T28a	SMI# hold time	5			5-6	Note 7
T29	PEREQ, ERROR#, BUSY# setup time	5		ns	5-6	Note 7
T30	PEREQ, ERROR#, BUSY# hold time	3			5-6	Note 7
T31	Clock-doubled PLL lock time		20	µs		Note 8

Notes: 1) V_{CC} = 4.75 V to 5.25 V, T_C = 0 to 100 °C
2) Input clock can be stopped; therefore, minimum CLK2 frequency is 0 MHz.
3) These parameters are not tested. They are determined by design characterization.
4) Float condition occurs when maximum output current becomes less than I_I in magnitude. Float is not 100% tested.
5) Not 100% tested.
6) SUSPA# floats only in response to activation of FLT#. SUSPA# does not float during a hold acknowledge state.
7) These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes to assure recognition within a specific CLK2 period.
8) Delay time from setting CKD in CCR0 to entering clock-doubled mode.

Table 5–23. AC Characteristics for TI486SXLC2–050 (See Note 1)

Symbol	Parameter	TI486SXLC2-050 MIN	TI486SXLC2-050 MAX	Unit	Figure	Notes
	CLK2 clock-doubled frequency range	32	50	MHz		
T1	CLK2 period	20		ns	5-4	Note 2
T2a	CLK2 high time	7			5-4	Note 3
T2b	CLK2 high time	4			5-4	Note 3
T3a	CLK2 low time	7			5-4	Note 3
T3b	CLK2 low time	5			5-4	Note 3
T4	CLK2 fall time		7		5-4	Note 3
T5	CLK2 rise time		7		5-4	Note 3
T6	A23–A1 valid delay	4	21	ns	5-7, 5-10	C_L = 50 pF
T6a	SMI# valid delay	4	30		5-7, 5-10	C_L = 50 pF
T7	A23–A1 float delay	4	30		5-10	Note 4
T8	BHE#, BLE#, LOCK# valid delay	4	21	ns	5-7, 5-10	C_L = 50 pF
T9	BHE#, BLE#, LOCK# float delay	4	30		5-10	Note 4
T10	ADS#, D/C#, M/IO#, W/R# valid delay	4	21	ns	5-7, 5-10	C_L = 50 pF
T10a	SMADS# valid delay	4	21		5-7, 5-10	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, W/R# float delay	4	30	ns	5-10	Note 4
T11a	SMADS# float delay	4	30		5-10	Note 4
T12	D15–D0 write data, SUSPA# valid delay	7	27	ns	5-7, 5-8	C_L = 50 pF,
T12a	D15–D0 write data hold time	2			5-9	Note 5
T13	D15–D0 write data, SUSPA# float delay	4	22		5-10	Notes 4, 6
T14	HDLA valid delay	4	22	ns	5-10	C_L = 50 pF
T15	NA#, SUSP#, FLUSH#, KEN#, A20M# setup time	5		ns	5-6	
T16	NA#, SUSP#, FLUSH#, KEN#, A20M# hold time	3			5-6	
T19	READY# setup time	9		ns	5-6	
T20	READY# hold time	4			5-6	
T21	D15–D0 read data setup time	7		ns	5-6	
T22	D15–D0 read data hold time	5			5-6	
T23	HOLD setup time	9		ns	5-6	
T24	HOLD hold time	3			5-6	
T25	RESET setup time	8		ns	5-5	
T26	RESET hold time	3			5-5	Note 5
T27	NMI, INTR setup time	6		ns	5-6	Note 7
T27a	SMI# setup time	6			5-6	Note 7
T28	NMI, INTR hold time	6			5-6	Note 7
T28a	SMI# hold time	6			5-6	Note 7
T29	PEREQ, ERROR#, BUSY# setup time	6		ns	5-6	Note 7
T30	PEREQ, ERROR#, BUSY# hold time	5			5-6	Note 7
T31	Clock-doubled PLL lock time		20	µs		Note 8

Notes:
1) V_{CC} = 4.75 V to 5.25 V, T_C = 0 to 100 °C
2) Input clock can be stopped; therefore, minimum CLK2 frequency is 0 MHz.
3) These parameters are not tested. They are determined by design characterization.
4) Float condition occurs when maximum output current becomes less than I_I in magnitude. Float is not 100% tested.
5) Not 100% tested.
6) SUSPA# floats only in response to activation of FLT#. SUSPA# does not float during a hold acknowledge state.
7) These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes to assure recognition within a specific CLK2 period.
8) Delay time from setting CKD in CCR0 to entering clock-doubled mode.

Table 5–24. AC Characteristics for TI486SXL-040 (See Note 1)

Symbol	Parameter	TI486SXL-040 MIN	TI486SXL-040 MAX	Unit	Figure	Notes
	CLK2 frequency range	32	40	MHz		
T1	CLK2 period	12.5		ns	5-4	Note 2
T2a	CLK2 high time	5			5-4	Note 3
T2b	CLK2 high time	3.25			5-4	Note 3
T3a	CLK2 low time	5			5-4	Note 3
T3b	CLK2 low time	3.25			5-4	Note 3
T4	CLK2 fall time		4		5-4	Note 3
T5	CLK2 rise time		4		5-4	Note 3
T6	A31–A2 valid delay	3	12.5	ns	5-12, 5-15	C_L = 50 pF
T6a	SMI# valid delay	3	12.5		5-12, 5-15	C_L = 50 pF
T7	A31–A2 float delay	3	17		5-15	Note 4
T8	BE3# – BE0#, LOCK# valid delay	3	12.5	ns	5-12, 5-15	C_L = 50 pF
T9	BE3# – BE0#, LOCK# float delay	3	17		5-15	Note 4
T10	ADS#, D/C#, M/IO#, W/R# valid delay	3	12.5	ns	5-12, 5-15	C_L = 50 pF
T10a	SMADS# valid delay	3	12.5		5-12, 5-15	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, W/R# float delay	3	17	ns	5-15	Note 4
T11a	SMADS# float delay	3	17		5-15	Note 4
T12	D31–D0 write data, SUSPA# valid delay	5	20	ns	5-12, 5-13	C_L = 50 pF,
T12a	D31–D0 write data hold time	2			5-14	Note 5
T13	D31–D0 write data, SUSPA# float delay	3	14.5		5-15	Notes 4, 6
T14	HDLA valid delay	3	17	ns	5-15	C_L = 50 pF
T15	A20M#, FLUSH#, KEN#, NA#, SUSP# setup time	5		ns	5-11	
T16	A20M#, FLUSH#, KEN#, NA#, SUSP# hold time	2			5-11	
T17	BS16# setup time	5		ns	5-11	
T18	BS16# hold time	2			5-11	
T19	READY# setup time	5		ns	5-11	
T20	READY# hold time	3			5-11	
T21	D31–D0 read data setup time	5		ns	5-11	
T22	D31–D0 read data hold time	3			5-11	
T23	HOLD setup time	4		ns	5-11	
T24	HOLD hold time	2			5-11	
T25	RESET setup time	4.5		ns	5-5	
T26	RESET hold time	2			5-5	Note 5
T27	NMI, INTR setup time	5		ns	5-10	Note 7
T27a	SMI# setup time	5			5-10	Note 7
T28	NMI, INTR hold time	5			5-10	Note 7
T28a	SMI# hold time	5			5-10	Note 7
T29	PEREQ, ERROR#, BUSY# setup time	5		ns	5-10	Note 7
T30	PEREQ, ERROR#, BUSY# hold time	3			5-10	Note 7
T31	Clock-doubled PLL lock time		20	µs		Note 8

Notes:
1) V_{CC} = 4.75 V to 5.25 V, for T_C see Table 5–6
2) Input clock can be stopped; therefore, minimum CLK2 frequency is 0 MHz.
3) These parameters are not tested. They are determined by design characterization.
4) Float condition occurs when maximum output current becomes less than I_I in magnitude. Float is not 100% tested.
5) Not 100% tested.
6) SUSPA# floats only in response to activation of FLT#. SUSPA# does not float during a hold-acknowledge state.
7) These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes to assure recognition within a specific CLK2 period.
8) Delay time from setting CKD in CCR0 to entering clock-doubled mode.

Table 5–25. AC Characteristics for TI486SXL2-050 (See Note 1)

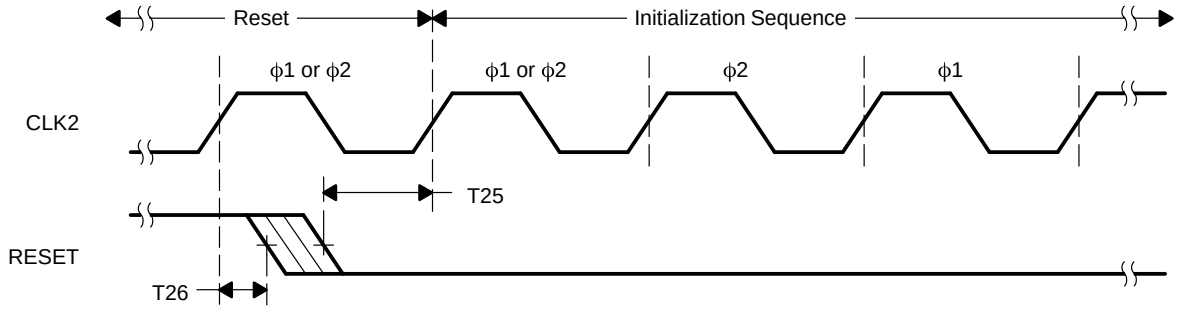
Symbol	Parameter	TI486SXL2-050 MIN	TI486SXL2-050 MAX	Unit	Figure	Notes
	CLK2 clock-doubled frequency range	32	50	MHz		
T1	CLK2 period	20		ns	5-4	Note 2
T2a	CLK2 high time	7			5-4	Note 3
T2b	CLK2 high time	4			5-4	Note 3
T3a	CLK2 low time	7			5-4	Note 3
T3b	CLK2 low time	5			5-4	Note 3
T4	CLK2 fall time		7		5-4	Note 3
T5	CLK2 rise time		7		5-4	Note 3
T6	A31–A2 valid delay	3	21	ns	5-12, 5-15	C_L = 50 pF
T6a	SMI# valid delay	3	30		5-12, 5-15	C_L = 50 pF
T7	A31–A2 float delay	4	30		5-15	Note 4
T8	BE3# – BE0#, LOCK# valid delay	2.5	18	ns	5-12, 5-15	C_L = 50 pF
T9	BE3# – BE0#, LOCK# float delay	4	30		5-15	Note 4
T10	ADS#, D/C#, M/IO#, W/R# valid delay	4	19	ns	5-12, 5-15	C_L = 50 pF
T10a	SMADS# valid delay	4	19		5-12, 5-15	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, W/R# float delay	4	30	ns	5-15	Note 4
T11a	SMADS# float delay	4	30		5-15	Note 4
T12	D31–D0 write data, SUSPA# valid delay	3.5	27	ns	5-12, 5-13	C_L = 50 pF,
T12a	D31–D0 write data hold time	2			5-14	Note 5
T13	D31–D0 write data, SUSPA# float delay	4	22		5-15	Notes 4, 6
T14	HDLA valid delay	2	22	ns	5-15	C_L = 50 pF
T15	A20M#, FLUSH#, KEN#, NA#, SUSP# setup time	5		ns	5-11	
T16	A20M#, FLUSH#, KEN#, NA#, SUSP# hold time	3.5			5-11	
T17	BS16# setup time	7		ns	5-11	
T18	BS16# hold time	2			5-11	
T19	READY# setup time	9		ns	5-11	
T20	READY# hold time	4			5-11	
T21	D31–D0 read data setup time	7		ns	5-11	
T22	D31–D0 read data hold time	5			5-11	
T23	HOLD setup time	9		ns	5-11	
T24	HOLD hold time	3.5			5-11	
T25	RESET setup time	8		ns	5-5	
T26	RESET hold time	3			5-5	Note 5
T27	NMI, INTR setup time	6		ns	5-10	Note 7
T27a	SMI# setup time	6			5-10	Note 7
T28	NMI, INTR hold time	6			5-10	Note 7
T28a	SMI# hold time	6			5-10	Note 7
T29	PEREQ, ERROR#, BUSY# setup time	6		ns	5-10	Note 7
T30	PEREQ, ERROR#, BUSY# hold time	5			5-10	Note 7
T31	Clock-doubled PLL lock time		20	µs		Note 8

Notes:
1) V_{CC} = 4.75 V to 5.25 V, for T_C see Table 5–6
2) Input clock can be stopped; therefore, minimum CLK2 frequency is 0 MHz.
3) These parameters are not tested. They are determined by design characterization.
4) Float condition occurs when maximum output current becomes less than I_I in magnitude. Float is not 100% tested.
5) Not 100% tested.
6) SUSPA# floats only in response to activation of FLT#. SUSPA# does not float during a hold-acknowledge state.
7) These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes to assure recognition within a specific CLK2 period.
8) Delay time from setting CKD in CCR0 to entering clock-doubled mode.

5.5.4 RESET Setup and Hold Timing

RESET setup and hold timing for the TI486SXL(C) family of microprocessors are illustrated in Figure 5–5.

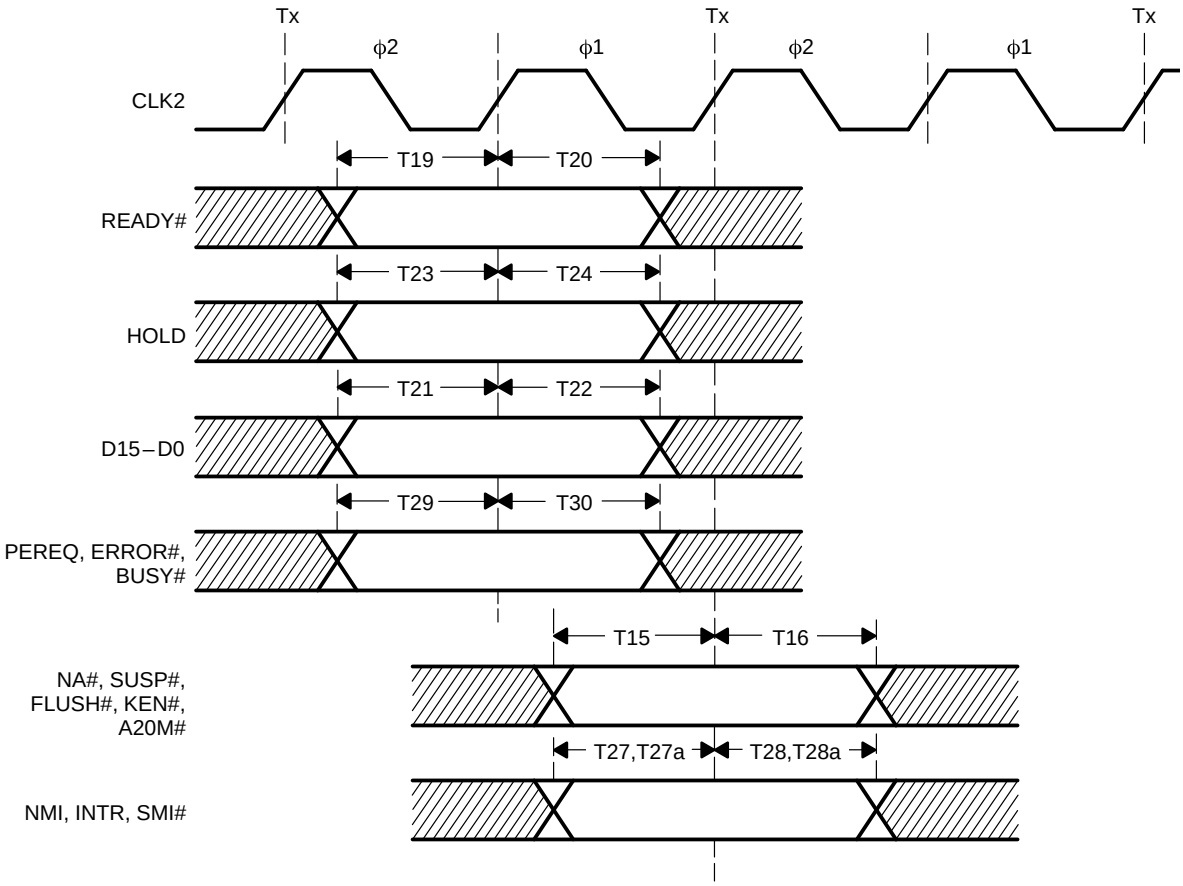
Figure 5–5. RESET Setup and Hold Timing



5.5.5 TI486SXLC Switching Waveforms

Switching waveforms for the TI486SXLC microprocessors are illustrated in Figure 5–6, Figure 5–7, Figure 5–8, Figure 5–9, and Figure 5–10 on pages 5-29 through 5-31.

Figure 5–6. TI486SXLC Input Signal Setup and Hold Timing



ADVANCE INFORMATION concerns new products in the sampling or preproduction phase of development. Characteristic data and other specifications are subject to change without notice.

Figure 5–7. TI486SXLC Output Signal Valid Delay Timing

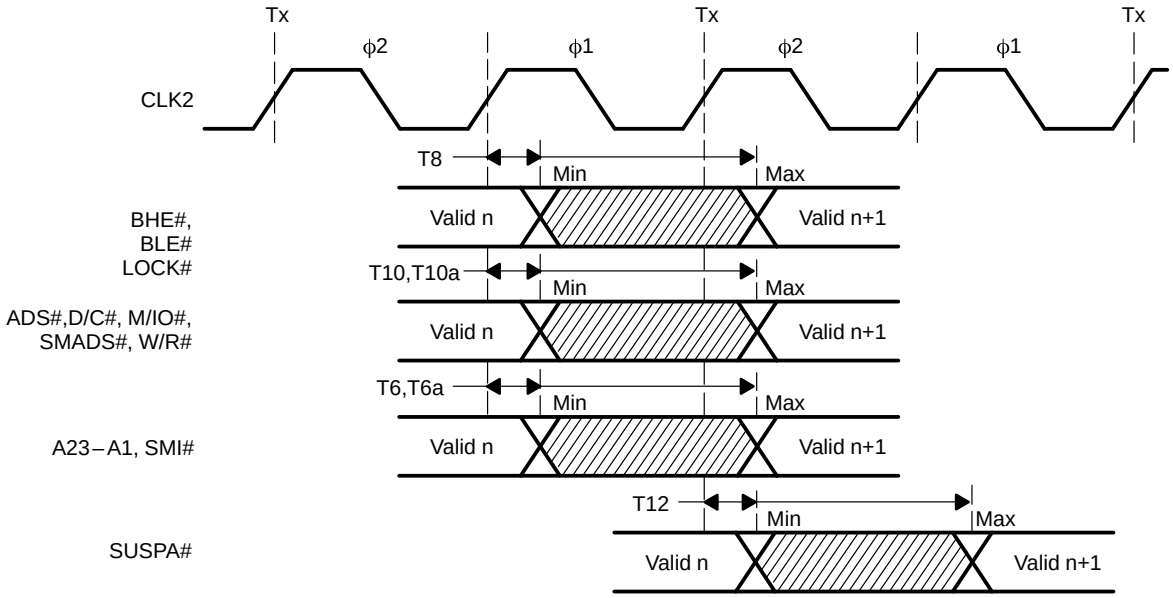


Figure 5–8. TI486SXLC Data Write Cycle Valid Delay Timing

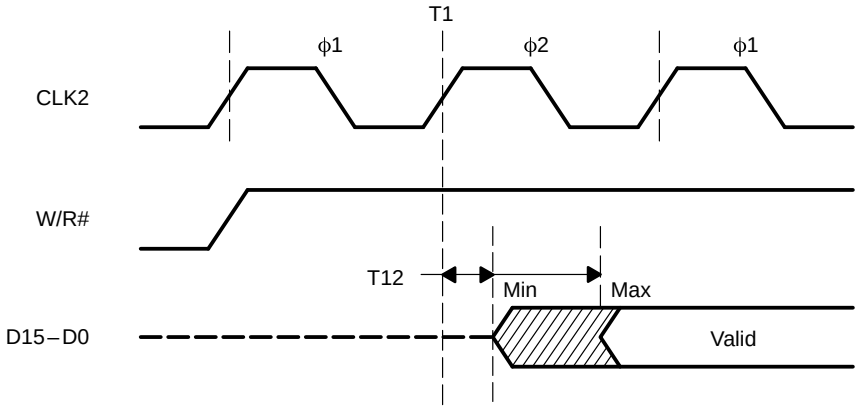


Figure 5–9. TI486SXLC Data Write Cycle Hold Timing

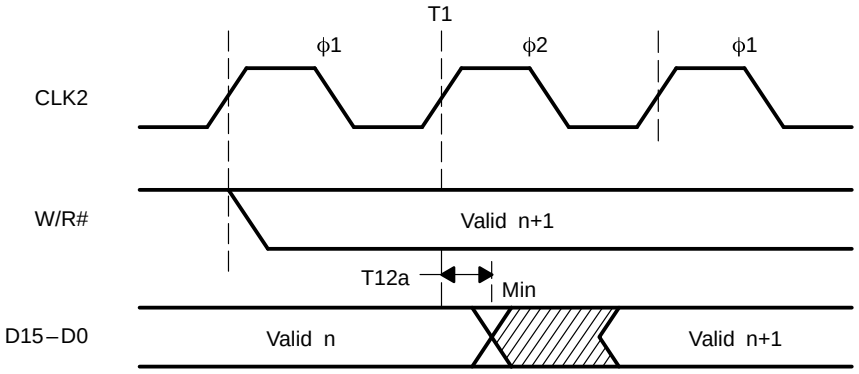
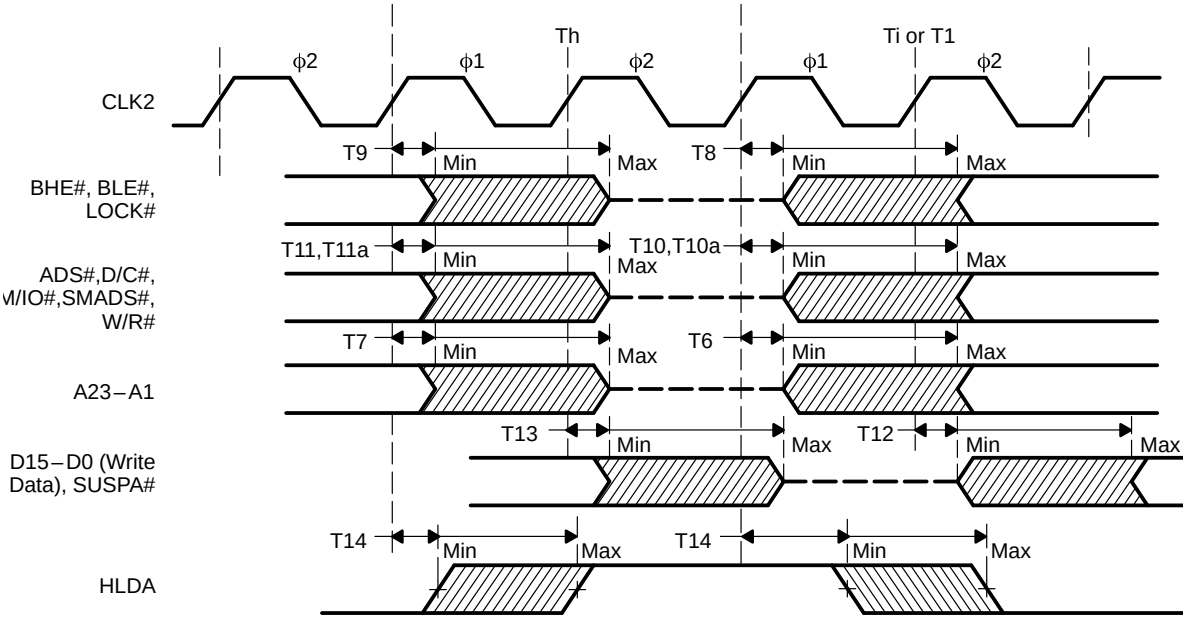


Figure 5–10. TI486SXLC Output Signal Float Delay and HLDA Valid Delay Timing



5.5.6 TI486SXL Switching Waveforms

Switching waveforms for the TI486SXL microprocessors are illustrated in Figure 5–11, Figure 5–12, Figure 5–13, Figure 5–14, and Figure 5–15 on pages 5-32 through 5-34.

Figure 5–11. TI486SXL Input Signal Setup and Hold Timing

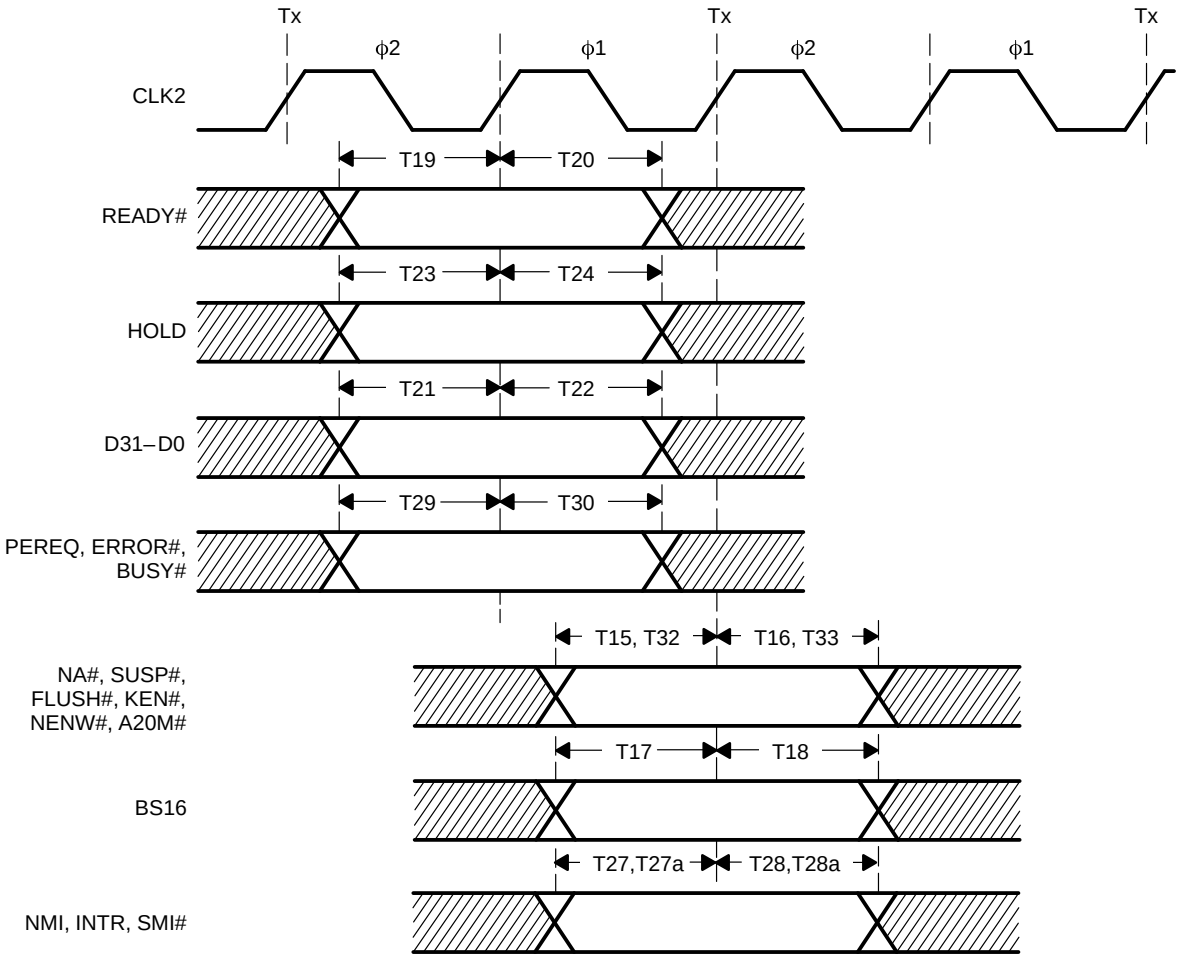


Figure 5–12. TI486SXL Output Signal Valid Delay Timing

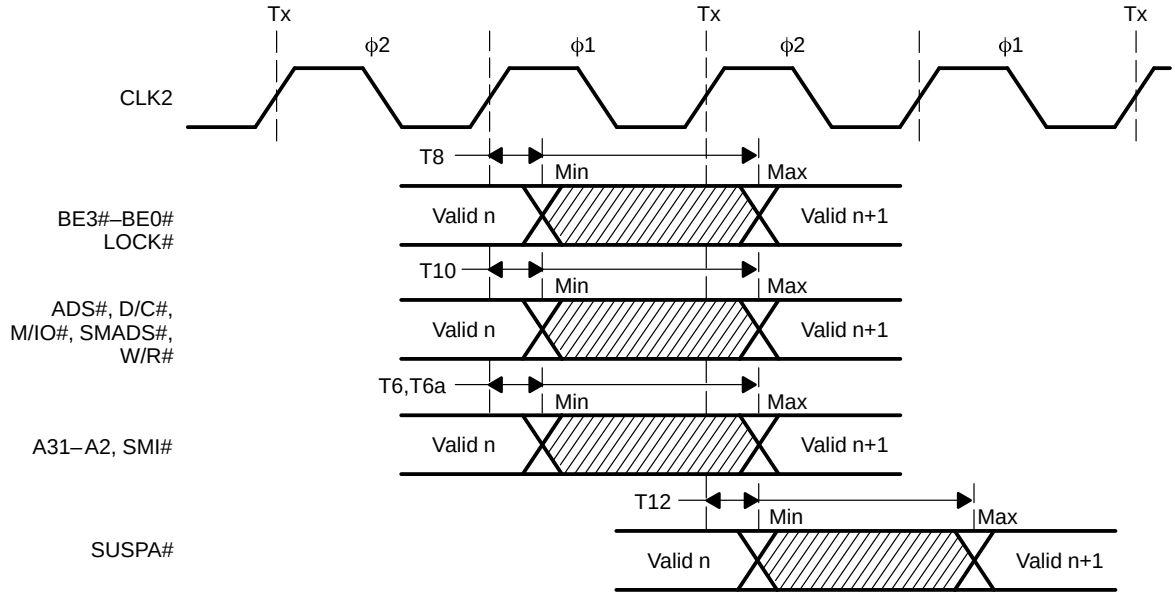


Figure 5–13. TI486SXL Data Write Cycle Valid Delay Timing

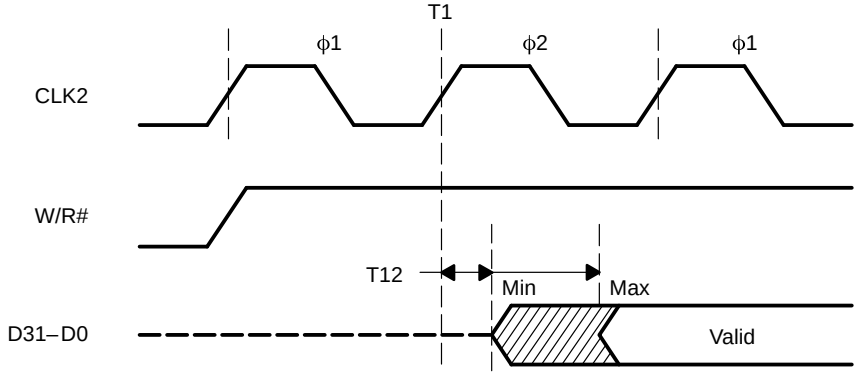


Figure 5–14. TI486SXL Data Write Cycle Hold Timing

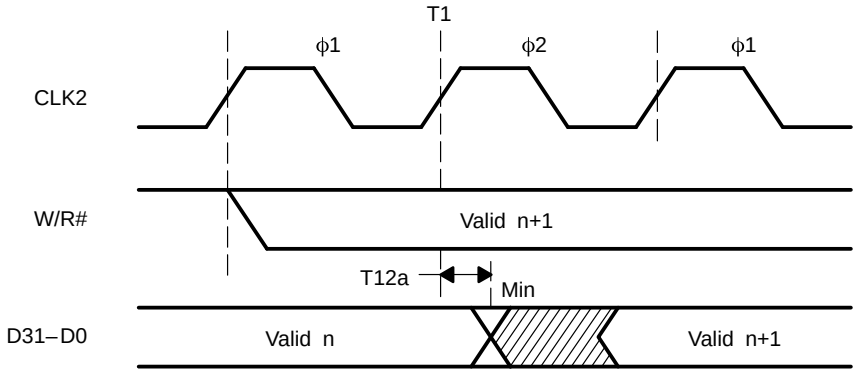
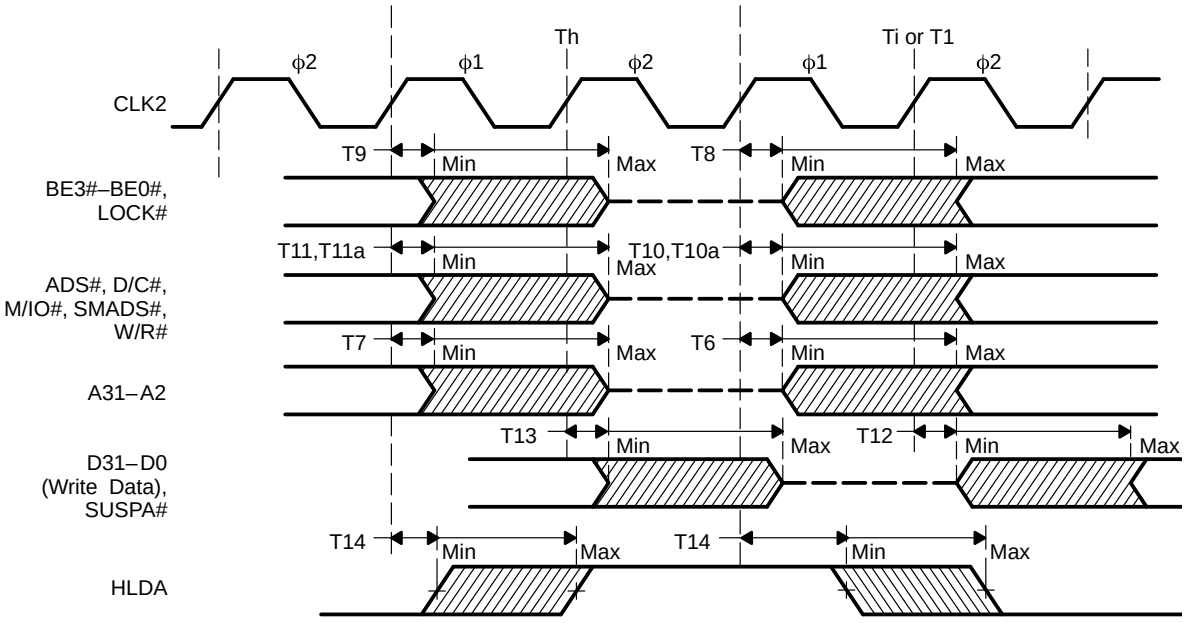


Figure 5–15. TI486SXL Output Signal Float Delay and HLDA Valid Delay Timing



Chapter 6

Mechanical Specifications

Mechanical specifications include pin assignments, package dimensions, and thermal characteristics for each of the TI486SXL(C) microprocessors.

The TI486SXL(C) microprocessors are supplied in the following packages:

- 100-pin, thermally enhanced plastic quad flat package
- 132-pin, ceramic pin grid array package
- 144-pin, thermally enhanced plastic quad flat package
- 144-pin, ceramic quad flat package
- 168-pin, ceramic pin grid array package

Pin assignments provide both a pin locator drawing and two pin listings. One pin listing is alphabetically by pin name and the other is (alpha)numerically by pin number.

A pinout cross-reference, comparing industry-standard 486SX pinouts, is supplied for the 168-pin package at the end of the pin-assignment data.

Industry-standard dimensioned drawings are supplied for each package.

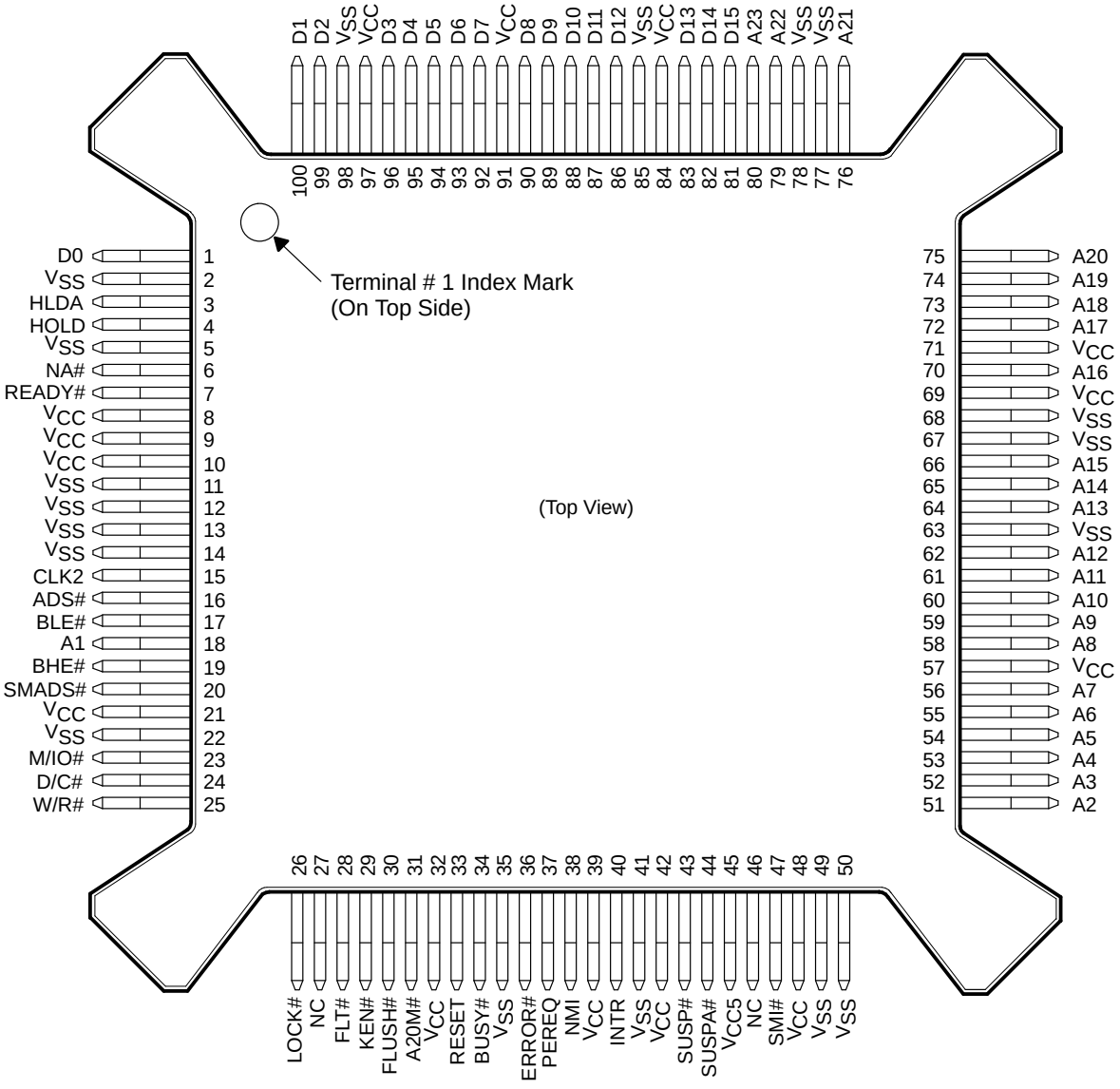
Thermal characteristics are supplied for each package and includes airflow measurement setup data for correlation purposes.

Topic **Page**

6.1 Terminal Assignments

The terminal assignments for the TI486SXLC microprocessors are shown in Figure 6–1. The signal names are shown in Table 6–1 sorted by terminal numbers and in Table 6–2 sorted by signal names.

Figure 6–1. TI486SXLC Terminal Assignments



NC — Make no external connection

Note:

Connecting or terminating (high or low) any NC terminal(s) may cause unpredictable results or nonperformance of the microprocessor.

Table 6–1. TI486SXLC Signal Names Sorted by Terminal Number

Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name
1	D0	21	V_{CC}	41	V_{SS}	61	A11	81	D15
2	V_{SS}	22	V_{SS}	42	V_{CC}	62	A12	82	D14
3	HLDA	23	M/IO#	43	SUSP#	63	V_{SS}	83	D13
4	HOLD	24	D/C#	44	SUSPA#	64	A13	84	V_{CC}
5	V_{SS}	25	W/R#	45	V_{CC5}	65	A14	85	V_{SS}
6	NA#	26	LOCK#	46	NC	66	A15	86	D12
7	READY#	27	NC	47	SMI#	67	V_{SS}	87	D11
8	V_{CC}	28	FLT#	48	V_{CC}	68	V_{SS}	88	D10
9	V_{CC}	29	KEN#	49	V_{SS}	69	V_{CC}	89	D9
10	V_{CC}	30	FLUSH#	50	V_{SS}	70	A16	90	D8
11	V_{SS}	31	A20M#	51	A2	71	V_{CC}	91	V_{CC}
12	V_{SS}	32	V_{CC}	52	A3	72	A17	92	D7
13	V_{SS}	33	RESET	53	A4	73	A18	93	D6
14	V_{SS}	34	BUSY#	54	A5	74	A19	94	D5
15	CLK2	35	V_{SS}	55	A6	75	A20	95	D4
16	ADS#	36	ERROR#	56	A7	76	A21	96	D3
17	BLE#	37	PEREQ	57	V_{CC}	77	V_{SS}	97	V_{CC}
18	A1	38	NMI	58	A8	78	V_{SS}	98	V_{SS}
19	BHE#	39	V_{CC}	59	A9	79	A22	99	D2
20	SMADS#	40	INTR	60	A10	80	A23	100	D1

Table 6–2. TI486SXLC Terminal Numbers Sorted by Signal Name

Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.
A1	18	A21	76	D11	87	PEREQ	37	V_{CC}	97
A2	51	A22	79	D12	86	READY#	7	V_{SS}	2
A3	52	A23	80	D13	83	RESET	33	V_{SS}	5
A4	53	ADS#	16	D14	82	SMADS#	20	V_{SS}	11
A5	54	A20M#	31	D15	81	SMI#	47	V_{SS}	12
A6	55	BHE#	19	D/C#	24	SUSP#	43	V_{SS}	13
A7	56	BLE#	17	ERROR#	36	SUSPA#	44	V_{SS}	14
A8	58	BUSY#	34	FLT#	28	V_{CC}	8	V_{SS}	22
A9	59	CLK2	15	FLUSH#	30	V_{CC}	9	V_{SS}	35
A10	60	D0	1	HOLD	4	V_{CC}	10	V_{SS}	41
A11	61	D1	100	HLDA	3	V_{CC}	21	V_{SS}	49
A12	62	D2	99	INTR	40	V_{CC}	32	V_{SS}	50
A13	64	D3	96	KEN#	29	V_{CC}	39	V_{SS}	63
A14	65	D4	95	LOCK#	26	V_{CC}	42	V_{SS}	67
A15	66	D5	94	M/IO#	23	V_{CC}	48	V_{SS}	68
A16	70	D6	93	NA#	6	V_{CC}	57	V_{SS}	77
A17	72	D7	92	NMI	38	V_{CC}	69	V_{SS}	78
A18	73	D8	90	NC	27	V_{CC}	71	V_{SS}	85
A19	74	D9	89	V_{CC5}	45	V_{CC}	84	V_{SS}	98
A20	75	D10	88	NC	46	V_{CC}	91	W/R#	25

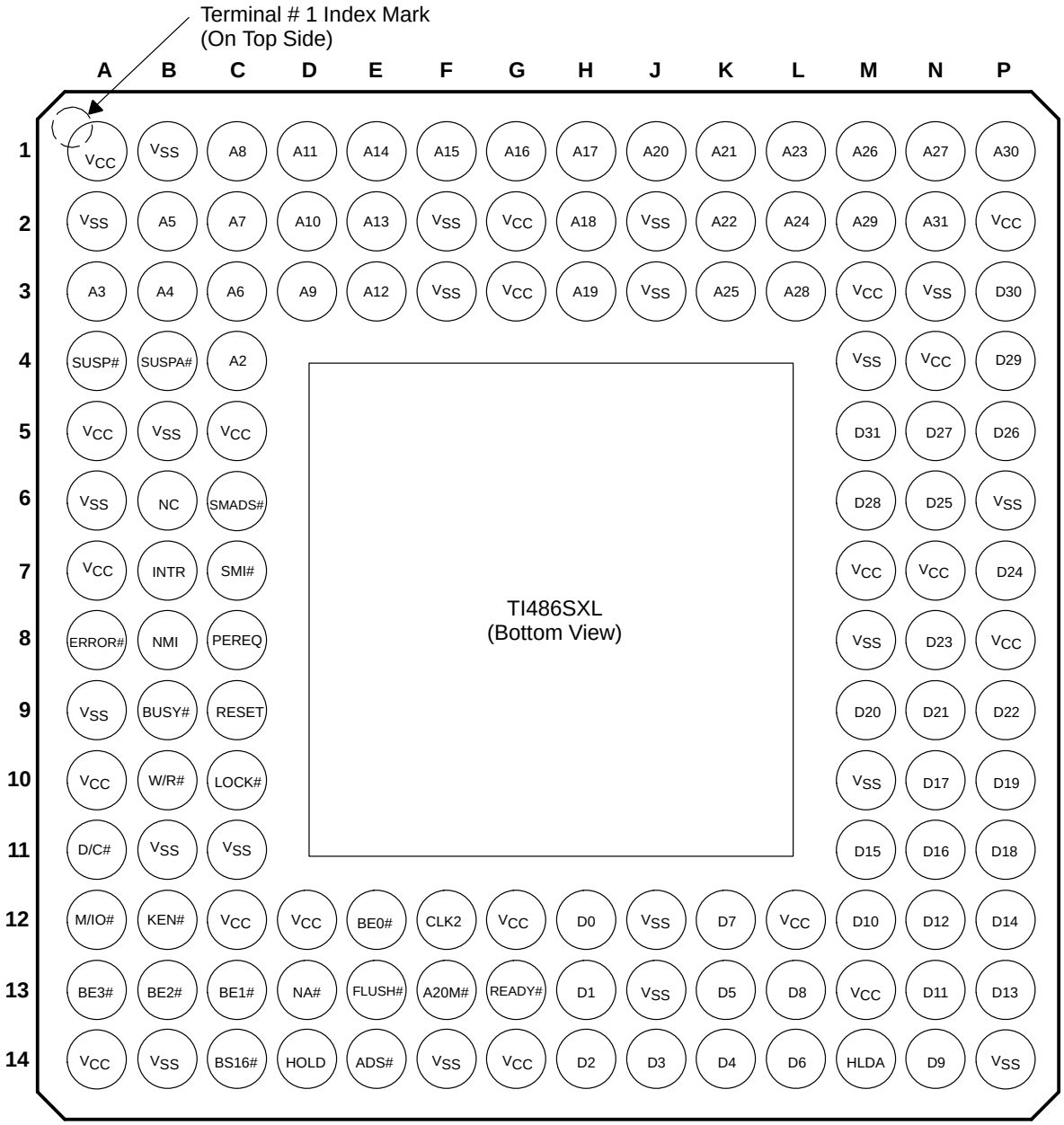
NC — Make no external connection

Note:

Connecting or terminating (high or low) any NC terminal(s) may cause unpredictable results or nonperformance of the microprocessor.

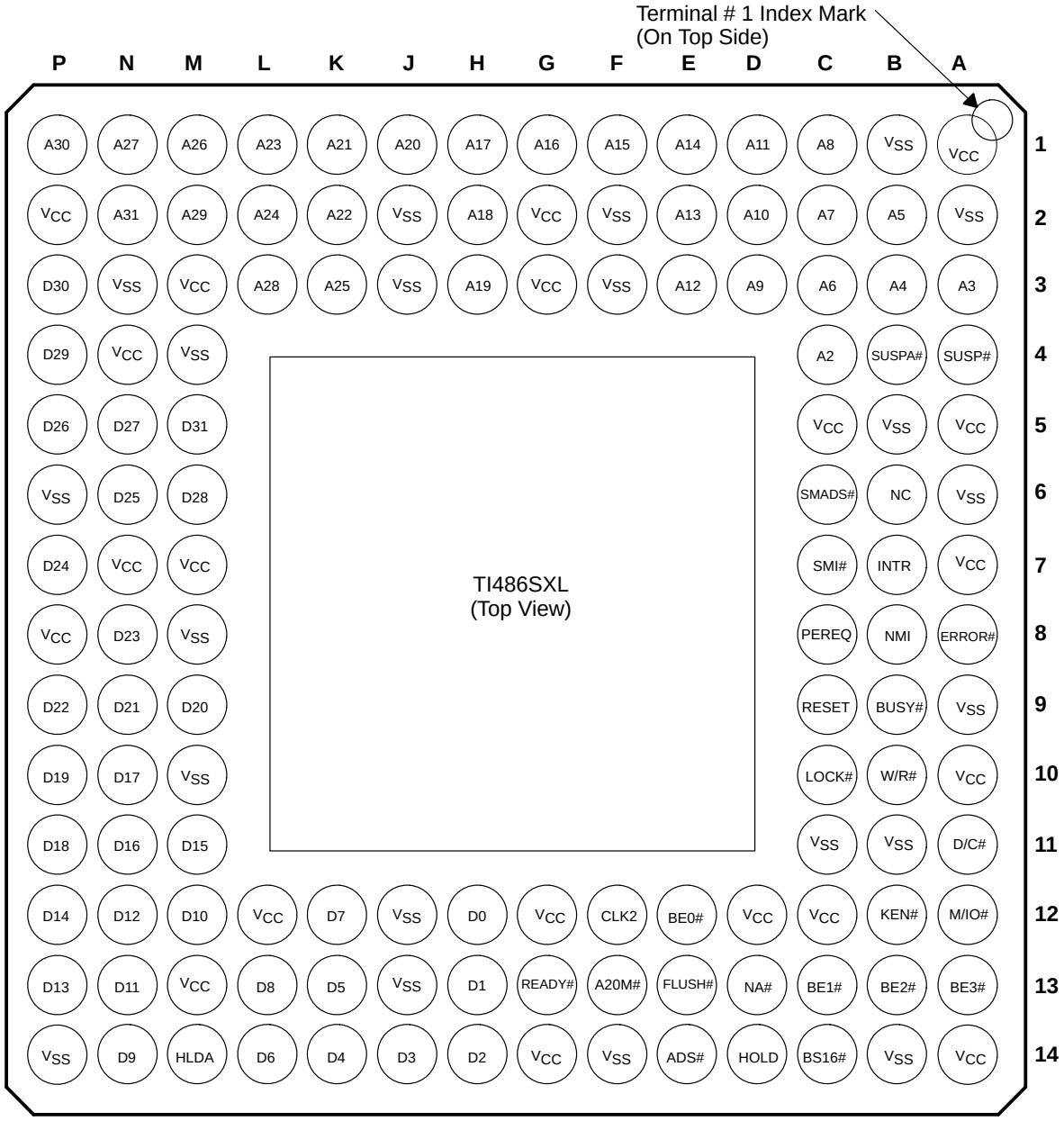
The terminal assignments for the 132-pin PGA TI486SXL microprocessors are shown as viewed from the terminal side (bottom) in Figure 6–2 and as viewed from the top side (component side when mounted on a PC board) in Figure 6–3. The signal names are listed in Table 6–3 and Table 6–4 sorted by terminal number and signal name, respectively.

Figure 6–2. 132-Pin PGA TI486SXL Package Terminals (Bottom View)



NC — Make no external connection

Figure 6–3. 132-Pin PGA TI486SXL Package Terminals (Top View)



NC — Make no external connection

Note:

Connecting or terminating (high or low) any NC terminal(s) may cause unpredictable results or nonperformance of the microprocessor.

Table 6–3. 132-Pin PGA TI486SXL Signal Names Sorted by Terminal Number

Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name
A1	V_{CC}	B9	BUSY#	D3	A9	H1	A17	L13	D8	N7	V_{CC}
A2	V_{SS}	B10	W/R#	D12	V_{CC}	H2	A18	L14	D6	N8	D23
A3	A3	B11	V_{SS}	D13	NA#	H3	A19	M1	A26	N9	D21
A4	SUSP#	B12	KEN#	D14	HOLD	H12	D0	M2	A29	N10	D17
A5	V_{CC}	B13	BE2#	E1	A14	H13	D1	M3	V_{CC}	N11	D16
A6	V_{SS}	B14	V_{SS}	E2	A13	H14	D2	M4	V_{SS}	N12	D12
A7	V_{CC}	C1	A8	E3	A12	J1	A20	M5	D31	N13	D11
A8	ERROR#	C2	A7	E12	BE0#	J2	V_{SS}	M6	D28	N14	D9
A9	V_{SS}	C3	A6	E13	FLUSH#	J3	V_{SS}	M7	V_{CC}	P1	A30
A10	V_{CC}	C4	A2	E14	ADS#	J12	V_{SS}	M8	V_{SS}	P2	V_{CC}
A11	D/C#	C5	V_{CC}	F1	A15	J13	V_{SS}	M9	D20	P3	D30
A12	M/IO#	C6	SMADS#	F2	V_{SS}	J14	D3	M10	V_{SS}	P4	D29
A13	BE3#	C7	SMI#	F3	V_{SS}	K1	A21	M11	D15	P5	D26
A14	V_{CC}	C8	PEREQ	F12	CLK2	K2	A22	M12	D10	P6	V_{SS}
B1	V_{SS}	C9	RESET	F13	A20M#	K3	A25	M13	V_{CC}	P7	D24
B2	A5	C10	LOCK#	F14	V_{SS}	K12	D7	M14	HLDA	P8	V_{CC}
B3	A4	C11	V_{SS}	G1	A16	K13	D5	N1	A27	P9	D22
B4	SUSPA#	C12	V_{CC}	G2	V_{CC}	K14	D4	N2	A31	P10	D19
B5	V_{SS}	C13	BE1#	G3	V_{CC}	L1	A23	N3	V_{SS}	P11	D18
B6	NC	C14	BS16#	G12	V_{CC}	L2	A24	N4	V_{CC}	P12	D14
B7	INTR	D1	A11	G13	READY#	L3	A28	N5	D27	P13	D13
B8	NMI	D2	A10	G14	V_{CC}	L12	V_{CC}	N6	D25	P14	V_{SS}

NC — Make no external connection

Table 6–4. 132-Pin PGA TI486SXL Terminal Numbers Sorted by Signal Name

Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.
A2	C4	A23	L1	D4	K14	D26	P5	SUSP#	A4	V_{SS}	A2
A3	A3	A24	L2	D5	K13	D27	N5	SUSPA#	B4	V_{SS}	A6
A4	B3	A25	K3	D6	L14	D28	M6	V_{CC}	A1	V_{SS}	A9
A5	B2	A26	M1	D7	K12	D29	P4	V_{CC}	A5	V_{SS}	B1
A6	C3	A27	N1	D8	L13	D30	P3	V_{CC}	A7	V_{SS}	B5
A7	C2	A28	L3	D9	N14	D31	M5	V_{CC}	A10	V_{SS}	B11
A8	C1	A29	M2	D10	M12	ERROR#	A8	V_{CC}	A14	V_{SS}	B14
A9	D3	A30	P1	D11	N13	FLUSH#	E13	V_{CC}	C5	V_{SS}	C11
A10	D2	A31	N2	D12	N12	HLDA	M14	V_{CC}	C12	V_{SS}	F2
A11	D1	ADS#	E14	D13	P13	HOLD	D14	V_{CC}	D12	V_{SS}	F3
A12	E3	BE0#	E12	D14	P12	INTR	B7	V_{CC}	G2	V_{SS}	F14
A13	E2	BE1#	C13	D15	M11	KEN#	B12	V_{CC}	G3	V_{SS}	J2
A14	E1	BE2#	B13	D16	N11	LOCK#	C10	V_{CC}	G12	V_{SS}	J3
A15	F1	BE3#	A13	D17	N10	M/IO#	A12	V_{CC}	G14	V_{SS}	J12
A16	G1	BS16#	C14	D18	P11	NA#	D13	V_{CC}	L12	V_{SS}	J13
A17	H1	BUSY#	B9	D19	P10	NMI	B8	V_{CC}	M3	V_{SS}	M4
A18	H2	CLK2	F12	D20	M9	NC	B6	V_{CC}	M7	V_{SS}	M8
A19	H3	D/C#	A11	D21	N9	PEREQ	C8	V_{CC}	M13	V_{SS}	M10
A20	J1	D0	H12	D22	P9	READY#	G13	V_{CC}	N4	V_{SS}	N3
A20M#	F13	D1	H13	D23	N8	RESET	C9	V_{CC}	N7	V_{SS}	P6
A21	K1	D2	H14	D24	P7	SMI#	C7	V_{CC}	P2	V_{SS}	P14
A22	K2	D3	J14	D25	N6	SMADS#	C6	V_{CC}	P8	W/R#	B10

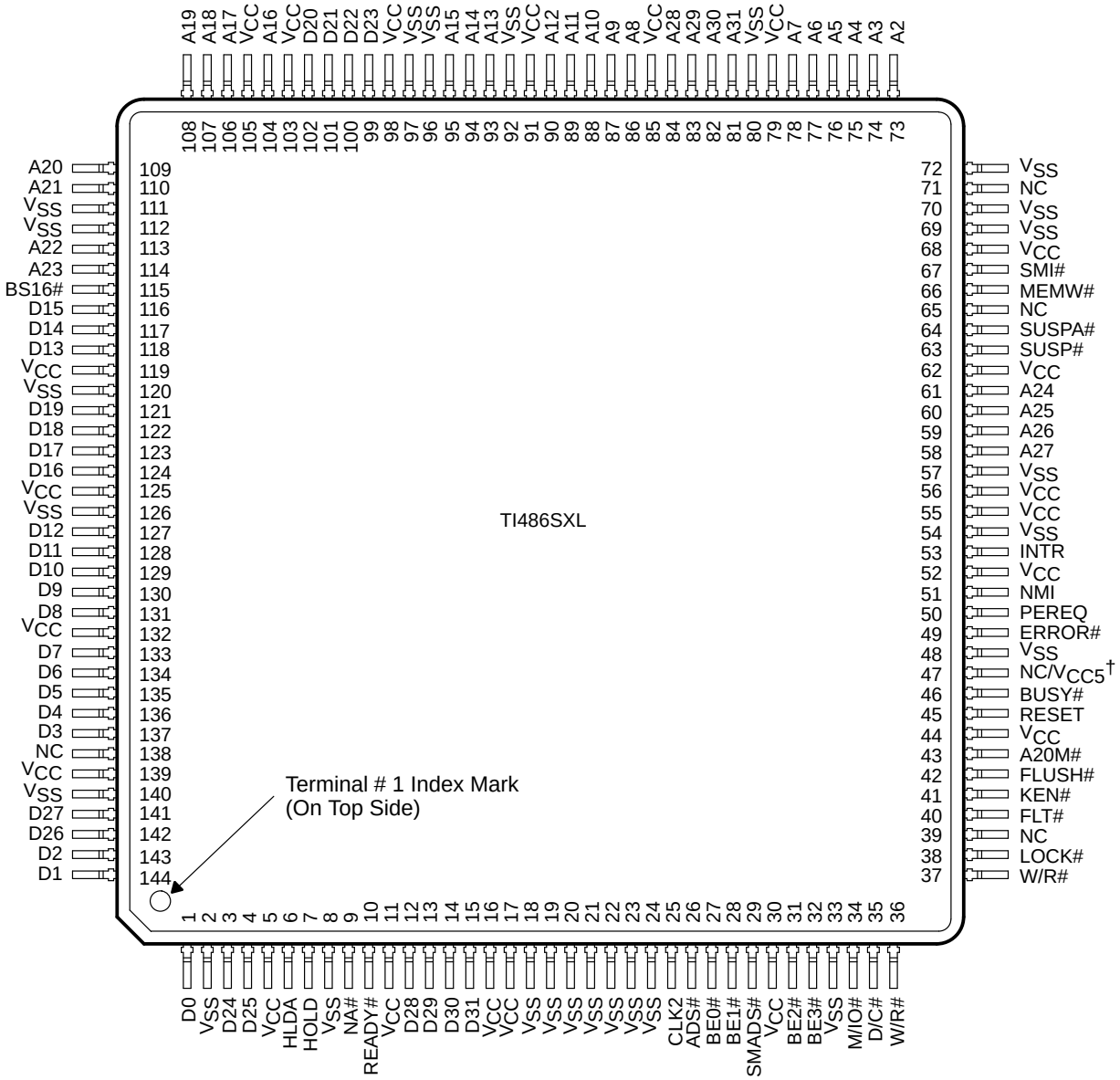
NC — Make no external connection

Note:

Connecting or terminating (high or low) any NC terminal(s) may cause unpredictable results or nonperformance of the microprocessor.

The terminal assignments for the 144-pin, QFP TI486SXL microprocessors are shown as viewed from the top side (component side when mounted on a PC board) in Figure 6–4. The signal names are listed in Table 6–5 and Table 6–6 sorted by terminal number and signal name, respectively.

Figure 6–4. 144-Pin QFP TI486SXL Package Terminals (Top View)



NC — Make no external connection

† This pin is V_{CC5} for the TI486SXL-G40 and TI486SXL2-G50. It is NC for all other devices.

Table 6–5. 144-Pin QFP TI486SXL Signal Names Sorted by Terminal Number

Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name
1	D0	25	CLK2	49	ERROR#	73	A2	97	V_{SS}	121	D19
2	V_{SS}	26	ADS#	50	PEREQ	74	A3	98	V_{CC}	122	D18
3	D24	27	BE0#	51	NMI	75	A4	99	D23	123	D17
4	D25	28	BE1#	52	V_{CC}	76	A5	100	D22	124	D16
5	V_{CC}	29	SMADS#	53	INTR	77	A6	101	D21	125	V_{CC}
6	HLDA	30	V_{CC}	54	V_{SS}	78	A7	102	D20	126	V_{SS}
7	HOLD	31	BE2#	55	V_{CC}	79	V_{CC}	103	V_{CC}	127	D2
8	V_{SS}	32	BE3#	56	V_{CC}	80	V_{SS}	104	A16	128	D11
9	NA#	33	V_{SS}	57	V_{SS}	81	A31	105	V_{CC}	129	D10
10	READY#	34	M/IO#	58	A27	82	A30	106	A17	130	D9
11	V_{CC}	35	D/C#	59	A26	83	A29	107	A18	131	D8
12	D28	36	W/R#	60	A25	84	A28	108	A19	132	V_{CC}
13	D29	37	W/R#	61	A24	85	V_{CC}	109	A20	133	D7
14	D30	38	LOCK#	62	V_{CC}	86	A8	110	A21	134	D6
15	D31	39	NC	63	SUSP#	87	A9	111	V_{SS}	135	D5
16	V_{CC}	40	FLT#	64	SUSPA#	88	A10	112	V_{SS}	136	D4
17	V_{CC}	41	KEN#	65	NC	89	A11	113	A22	137	D3
18	V_{SS}	42	FLUSH#	66	MEMW#	90	A12	114	A23	138	NC
19	V_{SS}	43	A20M#	67	SMI#	91	V_{CC}	115	BS16#	139	V_{CC}
20	V_{SS}	44	V_{CC}	68	V_{CC}	92	V_{SS}	116	D15	140	V_{SS}
21	V_{SS}	45	RESET	69	V_{SS}	93	A13	117	D14	141	D27
22	V_{SS}	46	BUSY#	70	V_{SS}	94	A14	118	D13	142	D26
23	V_{SS}	47	NC/V_{CC5}†	71	NC	95	A15	119	V_{CC}	143	D2
24	V_{SS}	48	V_{SS}	72	V_{SS}	96	V_{SS}	120	V_{SS}	144	D1

NC — Make no external connection
† This pin is V_{CC5} for the TI486SXL-G40 and TI486SXL2-G50. It is NC for all other devices.

Table 6–6. 144-Pin QFP TI486SXL Terminal Numbers Sorted by Signal Name

Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.
A2	73	A25	60	D8	131	ERROR#	49	V_{CC}	5	V_{SS}	19
A3	74	A26	59	D9	130	FLT#	40	V_{CC}	11	V_{SS}	20
A4	75	A27	58	D10	129	FLUSH#	42	V_{CC}	16	V_{SS}	21
A5	76	A28	84	D11	128	HLDA	6	V_{CC}	17	V_{SS}	22
A6	77	A29	83	D12	127	HOLD	7	V_{CC}	30	V_{SS}	23
A7	78	A30	82	D13	118	INTR	53	V_{CC}	44	V_{SS}	24
A8	86	A31	81	D14	117	KEN#	41	V_{CC}	52	V_{SS}	33
A9	87	ADS#	26	D15	116	LOCK#	38	V_{CC}	55	V_{SS}	48
A10	88	BE0#	27	D16	124	M/IO#	34	V_{CC}	56	V_{SS}	54
A11	89	BE1#	28	D17	123	MEMW#	66	V_{CC}	62	V_{SS}	57
A12	90	BE2#	31	D18	122	NA#	9	V_{CC}	68	V_{SS}	69
A13	93	BE3#	32	D19	121	NMI	51	V_{CC}	79	V_{SS}	70
A14	94	BS16#	115	D20	102	NC	39	V_{CC}	85	V_{SS}	72
A15	95	BUSY#	48	D21	101	NC/V_{CC5}†	47	V_{CC}	91	V_{SS}	80
A16	104	CLK2	25	D22	100	NC	65	V_{CC}	98	V_{SS}	92
A17	106	D/C#	35	D23	99	NC	71	V_{CC}	103	V_{SS}	96
A18	107	D0	1	D24	3	NC	138	V_{CC}	105	V_{SS}	97
A19	108	D1	144	D25	4	PEREQ	50	V_{CC}	119	V_{SS}	111
A20	109	D2	143	D26	142	READY#	10	V_{CC}	125	V_{SS}	112
A20M#	43	D3	137	D27	141	RESET	45	V_{CC}	132	V_{SS}	120
A21	110	D4	136	D28	12	SMI#	67	V_{CC}	139	V_{SS}	126
A22	113	D5	135	D29	13	SMADS#	29	V_{SS}	2	V_{SS}	140
A23	114	D6	134	D30	14	SUSP#	63	V_{SS}	8	W/R#	36
A24	61	D7	133	D31	15	SUSPA#	64	V_{SS}	18	W/R#	37

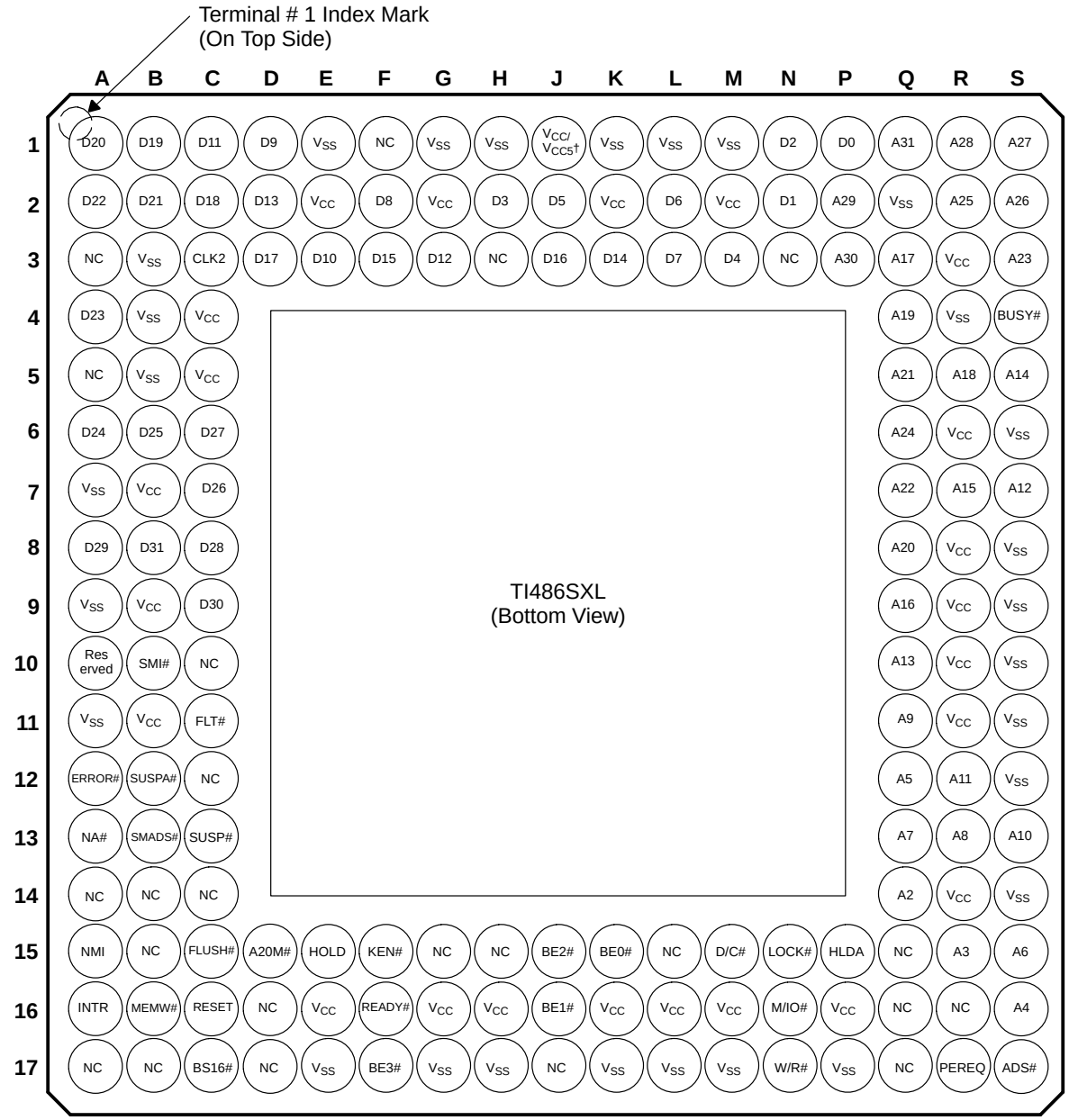
NC — Make no external connection
† This pin is V_{CC5} for the TI486SXL-G40 and TI486SXL2-G50. It is NC for all other devices.

Note:

Connecting or terminating (high or low) any NC terminal(s) may cause unpredictable results or nonperformance of the microprocessor.

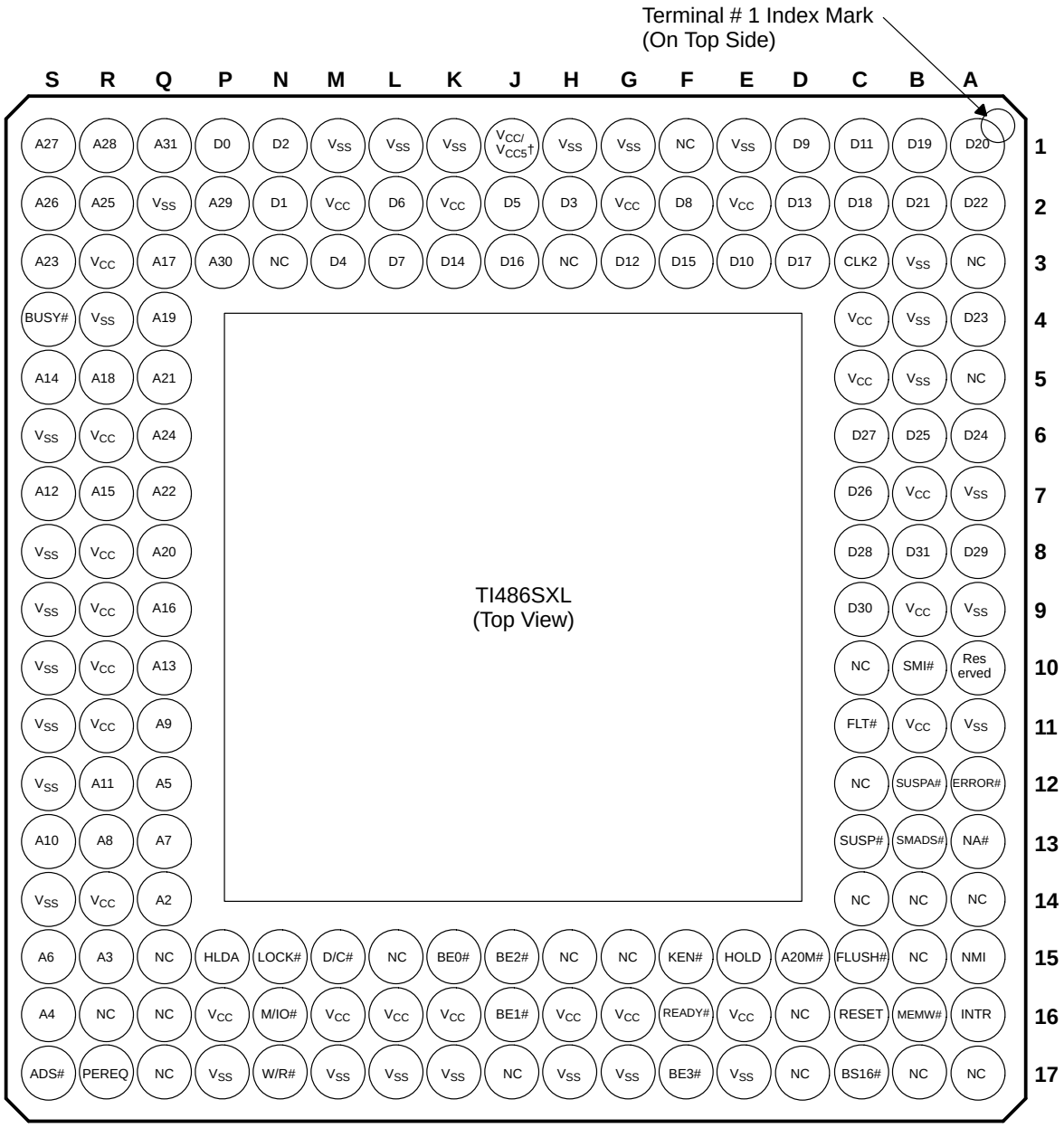
The terminal assignments for the 168-pin, PGA TI486SXL microprocessors are shown as viewed from the terminal side (bottom) in Figure 6–5 and as viewed from the top side (component side when mounted on a PC board) in Figure 6–6. The signal names are listed in Table 6–7 and Table 6–8 sorted by terminal number and signal name, respectively. In addition, Table 6–9 shows a cross-reference between the 168-pin TI486SXL pinout and the 486SX pinout.

Figure 6–5. 168-Pin PGA TI486SXL Package Terminals (Bottom View)



NC — Make no external connection
† This pin is V_{CC5} for the TI486SXL-G40 and TI486SXL2-G50. It is V_{CC} for all other devices.

Figure 6–6. 168-Pin PGA TI486SXL Package Terminals (Top View)



NC — Make no external connection
† This pin is V_{CC5} for the TI486SXL-G40 and TI486SXL2-G50. It is V_{CC} for all other devices.

Note:

Connecting or terminating (high or low) any NC terminal(s) may cause unpredictable results or nonperformance of the microprocessor.

Table 6–7. 168-Pin PGA TI486SXL Signal Names Sorted by Terminal Number

Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name
A1	D20	B12	SUSPA#	D17	NC	J15	BE2#	P2	A29	R7	A15
A2	D22	B13	SMADS#	E1	V_{SS}	J16	BE1#	P3	A30	R8	V_{CC}
A3	NC	B14	NC	E2	V_{CC}	J17	NC	P15	HLDA	R9	V_{CC}
A4	D23	B15	NC	E3	D10	K1	V_{SS}	P16	V_{CC}	R10	V_{CC}
A5	NC	B16	MEMW#	E15	HOLD	K2	V_{CC}	P17	V_{SS}	R11	V_{CC}
A6	D24	B17	NC	E16	V_{CC}	K3	D14	Q1	A31	R12	A11
A7	V_{SS}	C1	D11	E17	V_{SS}	K15	BE0#	Q2	V_{SS}	R13	A8
A8	D29	C2	D18	F1	NC	K16	V_{CC}	Q3	A17	R14	V_{CC}
A9	V_{SS}	C3	CLK2	F2	D8	K17	V_{SS}	Q4	A19	R15	A3
A10	Reserved	C4	V_{CC}	F3	D15	L1	V_{SS}	Q5	A21	R16	NC
A11	V_{SS}	C5	V_{CC}	F15	KEN#	L2	D6	Q6	A24	R17	PEREQ
A12	ERROR#	C6	D27	F16	READY#	L3	D7	Q7	A22	S1	A27
A13	NA#	C7	D26	F17	BE3#	L15	NC	Q8	A20	S2	A26
A14	NC	C8	D28	G1	V_{SS}	L16	V_{CC}	Q9	A16	S3	A23
A15	NMI	C9	D30	G2	V_{CC}	L17	V_{SS}	Q10	A13	S4	BUSY#
A16	INTR	C10	NC	G3	D12	M1	V_{SS}	Q11	A9	S5	A14
A17	NC	C11	FLT#	G15	NC	M2	V_{CC}	Q12	A5	S6	V_{SS}
B1	D19	C12	NC	G16	V_{CC}	M3	D4	Q13	A7	S7	A12
B2	D21	C13	SUSP#	G17	V_{SS}	M15	D/C#	Q14	A2	S8	V_{SS}
B3	V_{SS}	C14	NC	H1	V_{SS}	M16	V_{CC}	Q15	NC	S9	V_{SS}
B4	V_{SS}	C15	FLUSH#	H2	D3	M17	V_{SS}	Q16	NC	S10	V_{SS}
B5	V_{SS}	C16	RESET	H3	NC	N1	D2	Q17	NC	S11	V_{SS}
B6	D25	C17	BS16#	H15	NC	N2	D1	R1	A28	S12	V_{SS}
B7	V_{CC}	D1	D9	H16	V_{CC}	N3	NC	R2	A25	S13	A10
B8	D31	D2	D13	H17	V_{SS}	N15	LOCK#	R3	V_{CC}	S14	VSS
B9	V_{CC}	D3	D17	J1	$V_{CC(5†)}$	N16	M/IO#	R4	V_{SS}	S15	A6
B10	SMI#	D15	A20M#	J2	D5	N17	W/R#	R5	A18	S16	A4
B11	V_{CC}	D16	NC	J3	D16	P1	D0	R6	V_{CC}	S17	ADS#

Table 6–8. 168-Pin PGA TI486SXL Terminal Numbers Sorted by Signal Name

Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.	Signal Name	Term. No.
A2	Q14	A29	P2	D16	J3	NC	A3	SMADS#	B13	V_{SS}	A9
A3	R15	A30	P3	D17	D3	NC	A5	SUSP#	C13	V_{SS}	A11
A4	S16	A31	Q1	D18	C2	NC	A14	SUSPA#	B12	V_{SS}	B3
A5	Q12	ADS#	S17	D19	B1	NC	A17	V_{CC}	B7	V_{SS}	B4
A6	S15	BE0#	K15	D20	A1	NC	B14	V_{CC}	B9	V_{SS}	B5
A7	Q13	BE1#	J16	D21	B2	NC	B15	V_{CC}	B11	V_{SS}	E1
A8	R13	BE2#	J15	D22	A2	NC	B17	V_{CC}	C4	V_{SS}	E17
A9	Q11	BE3#	F17	D23	A4	NC	C10	V_{CC}	C5	V_{SS}	G1
A10	S13	BS16#	C17	D24	A6	NC	C12	V_{CC}	E2	V_{SS}	G17
A11	R12	BUSY#	S4	D25	B6	NC	C14	V_{CC}	E16	V_{SS}	H1
A12	S7	CLK2	C3	D26	C7	NC	D16	V_{CC}	G2	V_{SS}	H17
A13	Q10	D/C#	M15	D27	C6	NC	D17	V_{CC}	G16	V_{SS}	K1
A14	S5	D0	P1	D28	C8	NC	F1	V_{CC}	H16	V_{SS}	K17
A15	R7	D1	N2	D29	A8	NC	G15	$V_{CC(5†)}$	J1	V_{SS}	L1
A16	Q9	D2	N1	D30	C9	NC	H3	V_{CC}	K2	V_{SS}	L17
A17	Q3	D3	H2	D31	B8	NC	H15	V_{CC}	K16	V_{SS}	M1
A18	R5	D4	M3	ERROR#	A12	NC	J17	V_{CC}	L16	V_{SS}	M17
A19	Q4	D5	J2	FLT#	C11	NC	L15	V_{CC}	M2	V_{SS}	P17
A20	Q8	D6	L2	FLUSH#	C15	NC	N3	V_{CC}	M16	V_{SS}	Q2
A20M#	D15	D7	L3	HLDA	P15	NC	Q15	V_{CC}	P16	V_{SS}	R4
A21	Q5	D8	F2	HOLD	E15	NC	Q16	V_{CC}	R3	V_{SS}	S6
A22	Q7	D9	D1	INTR	A16	NC	Q17	V_{CC}	R6	V_{SS}	S8
A23	S3	D10	E3	KEN#	F15	NC	R16	V_{CC}	R8	V_{SS}	S9
A24	Q6	D11	C1	LOCK#	N15	PEREQ	R17	V_{CC}	R9	V_{SS}	S10
A25	R2	D12	G3	M/IO#	N16	READY#	F16	V_{CC}	R10	V_{SS}	S11
A26	S2	D13	D2	MEMW#	B16	Reserved	A10	V_{CC}	R11	V_{SS}	S12
A27	S1	D14	K3	NA#	A13	RESET	C16	V_{CC}	R14	V_{SS}	S14
A28	R1	D15	F3	NMI	A15	SMI#	B10	V_{SS}	A7	W/R#	N17

NC — Make no external connection

† This pin is V_{CC5} for the TI486SXL-G40 and TI486SXL2-G50. It is V_{CC} for all other devices.

Table 6–9. TI486SXL Signal Summary for 168-Pin PGA Pinout

Address			Data			Control			Miscellaneous and Spares			V_{CC}/V_{SS}		
486SX	486SXL	Pin	486SX	486SXL	Pin	486SX	486SXL	Pin	486SX	486SXL	Pin	486SX	486SXL	Pin
A2	A2	Q14	D0	D0	P1	A20M#	A20M#	D15	CLKSEL(LP)	NC	A3	V_{CC}	V_{CC}	B7, B9
A3	A3	R15	D1	D1	N2	ADS#	ADS#	S17	Reserved	Reserved	A10	V_{CC}	V_{CC}	B11, C4
A4	A4	S16	D2	D2	N1	AHOLD	NC	A17	NC	ERROR#	A12	V_{CC}	V_{CC}	C5, E2
A5	A5	Q12	D3	D3	H2	BE0#	BE0#	K15	NC	NA#	A13	V_{CC}	V_{CC}	E16, G2
A6	A6	S15	D4	D4	M3	BE1#	BE1#	J16	TDI(S/DX)	NC	A14	V_{CC}	V_{CC}	G16, H16
A7	A7	Q13	D5	D5	J2	BE2#	BE2#	J15	SMI#(S)	SMI#	B10	V_{CC5}(DX4)	$V_{CC(5\dagger)}$	J1
A8	A8	R13	D6	D6	L2	BE3#	BE3#	F17	NC	SUSPA#	B12	V_{CC}	V_{CC}	K2
A9	A9	Q11	D7	D7	L3	BLAST#	NC	R16	NC	SMADS#	B13	V_{CC}	V_{CC}	K16, L16
A10	A10	S13	D8	D8	F2	BOFF#	NC	D17	TMS	NC	B14	V_{CC}	V_{CC}	M2, M16
A11	A11	R12	D9	D9	D1	BRDY#	NC	H15	NMI(DX)	NC	B15	V_{CC}	V_{CC}	P16, R3
A12	A12	S7	D10	D10	E3	BREQ#	NC	Q15	TDO(S/DX)	MEMW#	B16	V_{CC}	V_{CC}	R6, R8
A13	A13	Q10	D11	D11	C1	BS8#	NC	D16	SRESET(S)	NC	C10	V_{CC}	V_{CC}	R9, R10
A14	A14	S5	D12	D12	G3	BS16#	BS16#	C17	UP#(S)	FLT#	C11	V_{CC}	V_{CC}	R11, R14
A15	A15	R7	D13	D13	D2	CLK	CLK2	C3	SMIACT#(S)	NC	C12			
A16	A16	Q9	D14	D14	K3	D/C#	D/C#	M15	NC	SUSP#	C13			
A17	A17	Q3	D15	D15	F3	DP0	NC	N3	FERR#(DX)	NC	C14	V_{SS}	V_{SS}	A7, A9
A18	A18	R5	D16	D16	J3	DP1	NC	F1	STPCLK(S)	NC	G15	V_{SS}	V_{SS}	A11, B3
A19	A19	Q4	D17	D17	D3	DP2	NC	H3	NC	PEREQ	R17	V_{SS}	V_{SS}	B4, B5
A20	A20	Q8	D18	D18	C2	DP3	NC	A5	NC	BUSY#	S4	V_{SS}	V_{SS}	E1, E17
A21	A21	Q5	D19	D19	B1	EADS#	NC	B17				V_{SS}	V_{SS}	G1, G17
A22	A22	Q7	D20	D20	A1	FLUSH#	FLUSH#	C15				V_{SS}	V_{SS}	H1, H17
A23	A23	S3	D21	D21	B2	HLDA	HLDA	P15				V_{SS}	V_{SS}	K1, K17
A24	A24	Q6	D22	D22	A2	HOLD	HOLD	E15				V_{SS}	V_{SS}	L1, L17
A25	A25	R2	D23	D23	A4	INTR	INTR	A16				V_{SS}	V_{SS}	M1, M17
A26	A26	S2	D24	D24	A6	KEN#	KEN#	F15				V_{SS}	V_{SS}	P17, Q2
A27	A27	S1	D25	D25	B6	LOCK#	LOCK#	N15				V_{SS}	V_{SS}	R4, S6
A28	A28	R1	D26	D26	C7	M/IO#	M/IO#	N16				V_{SS}	V_{SS}	S8, S9
A29	A29	P2	D27	D27	C6	NMI	NMI	A15				V_{SS}	V_{SS}	S10, S11
A30	A30	P3	D28	D28	C8	PCD	NC	J17				V_{SS}	V_{SS}	S12, S14
A31	A31	Q1	D29	D29	A8	PCHK#	NC	Q17						
			D30	D30	C9	PWT	NC	L15						
			D31	D31	B8	PCLOK#	NC	Q16						
						RDY#	READY#	F16						
						RESET	RESET	C16						
						W/R#	W/R#	N17						

(LP) = Low Power. (S) = 486SX, (DX) = 486DX, and (DX4) = 486DX4

† This pin is V_{CC5} for the TI486SXL-G40 and TI486SXL2-G50. It is V_{CC} for all other devices.

6.2 Package Dimensions

Figure 6–7 shows the package dimensions for the 100-pin TI486SXLC microprocessor.

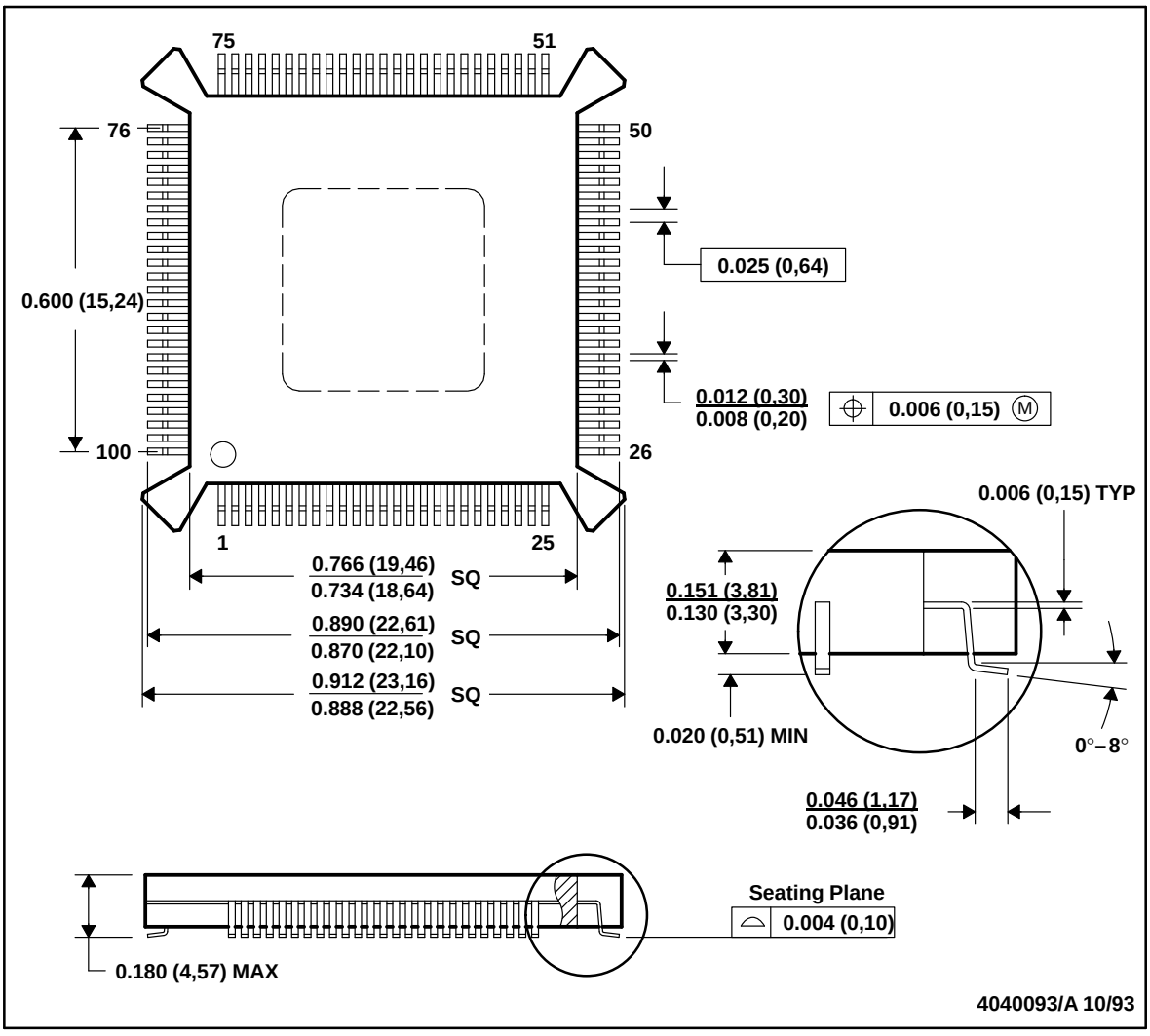
Figure 6–8 shows the package dimensions for the132-pin PGA TI486SXL.

Figure 6–9 and Figure 6–10 show the package dimensions for the 144-pin QFP TI486SXL.

Figure 6–11 shows the package dimensions for the 168-pin PGA TI486SXL.

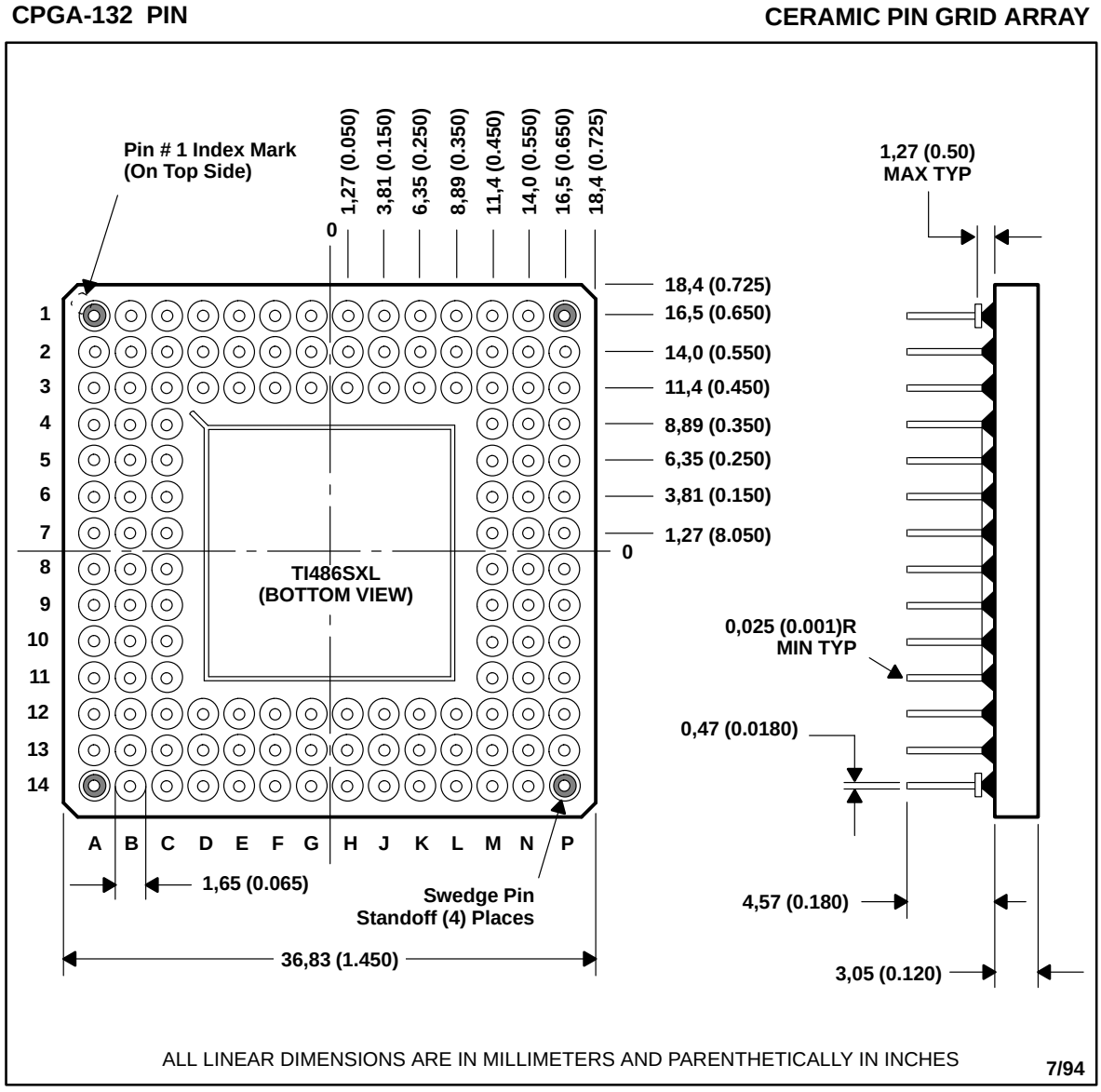
Figure 6–7. 100-Pin Thermally Enhanced Plastic QFP Package Dimensions (TI486SXLC)

PJF(S-PQFP-G100) **PLASTIC QUAD FLATPACK**



NOTES: A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Falls within JEDEC MO-069
D. Thermally enhanced molded plastic package with a heat slug (HSL) exposed on bottom side of the package body.

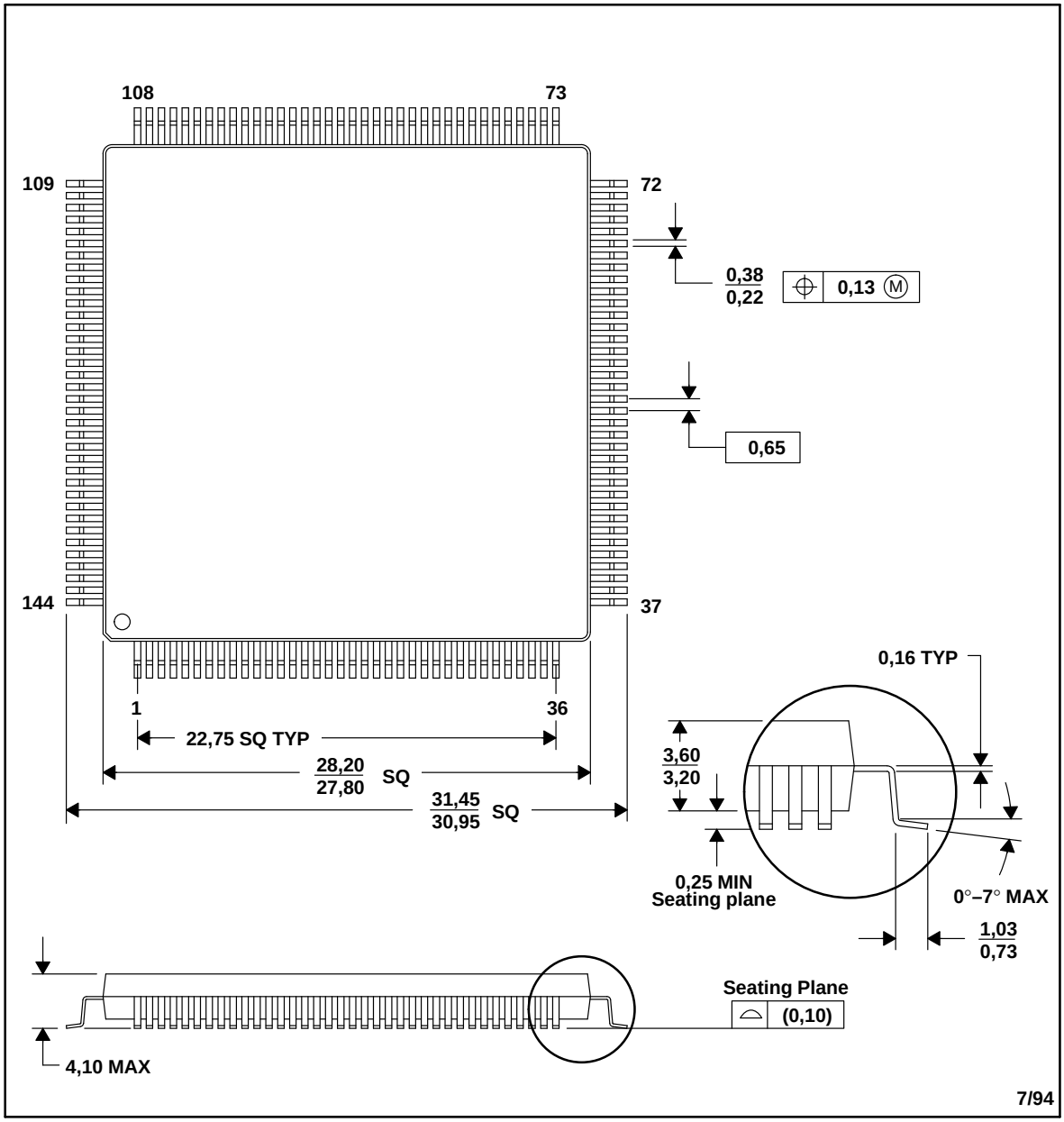
Figure 6–8. 132-Pin Ceramic PGA Package Dimensions (TI486SXL)



NOTES: E. All linear dimensions are in millimeters (inches).
F. This drawing is subject to change without notice.

Figure 6–9. 144-Pin Plastic QFP Dimensions (TI486SXL)

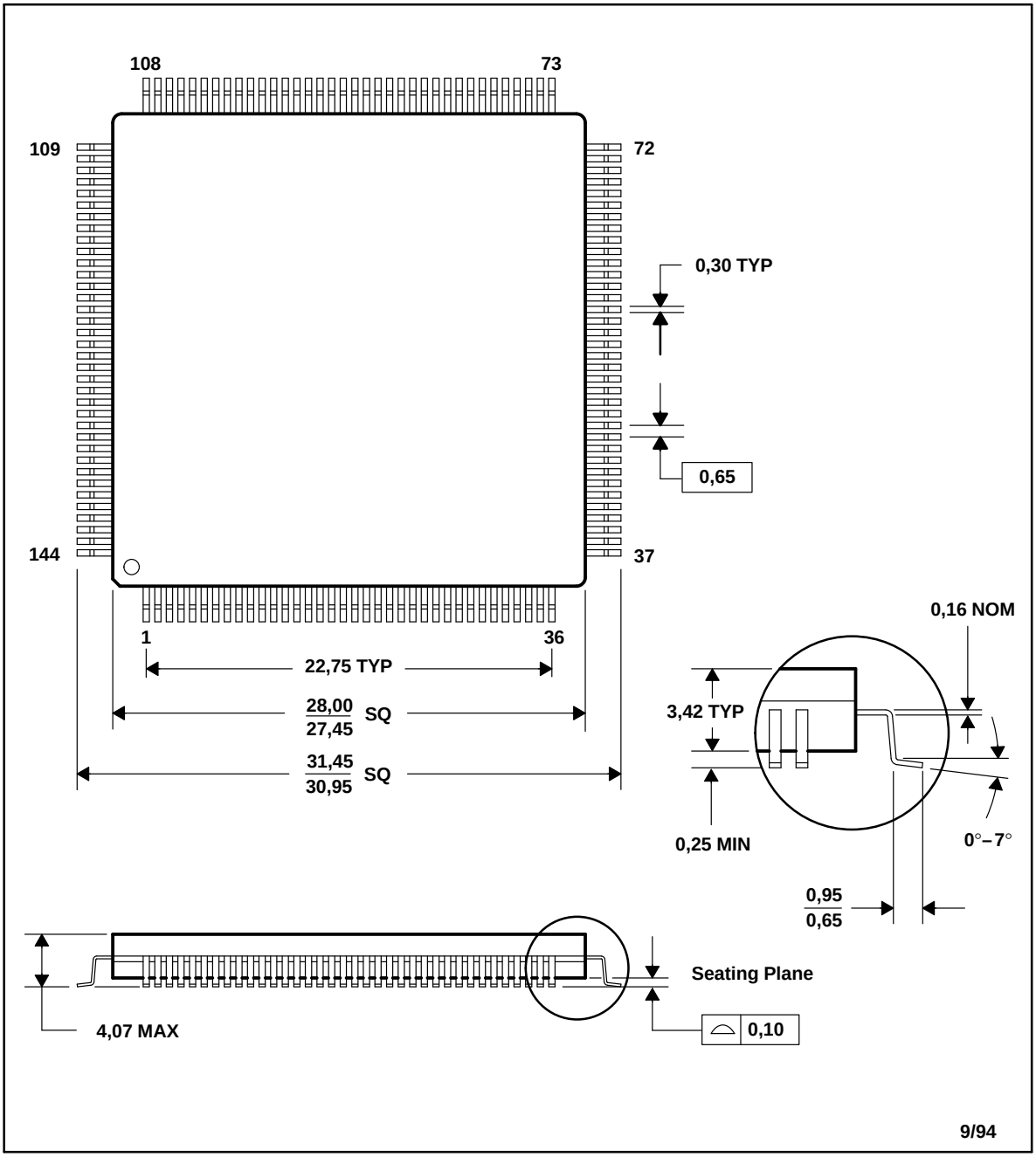
PCE(S-PQFP–G144) **PLASTIC QUAD FLATPACK**



NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS-022
D. Thermally enhanced molded plastic package with a heat spreader (HSP)
E. Foot length is measured from lead tip to a position on backside of lead 0,25mm above seating plane (gage plane).

Figure 6–10. 144-Pin Ceramic QFP Dimensions (TI486SXL)

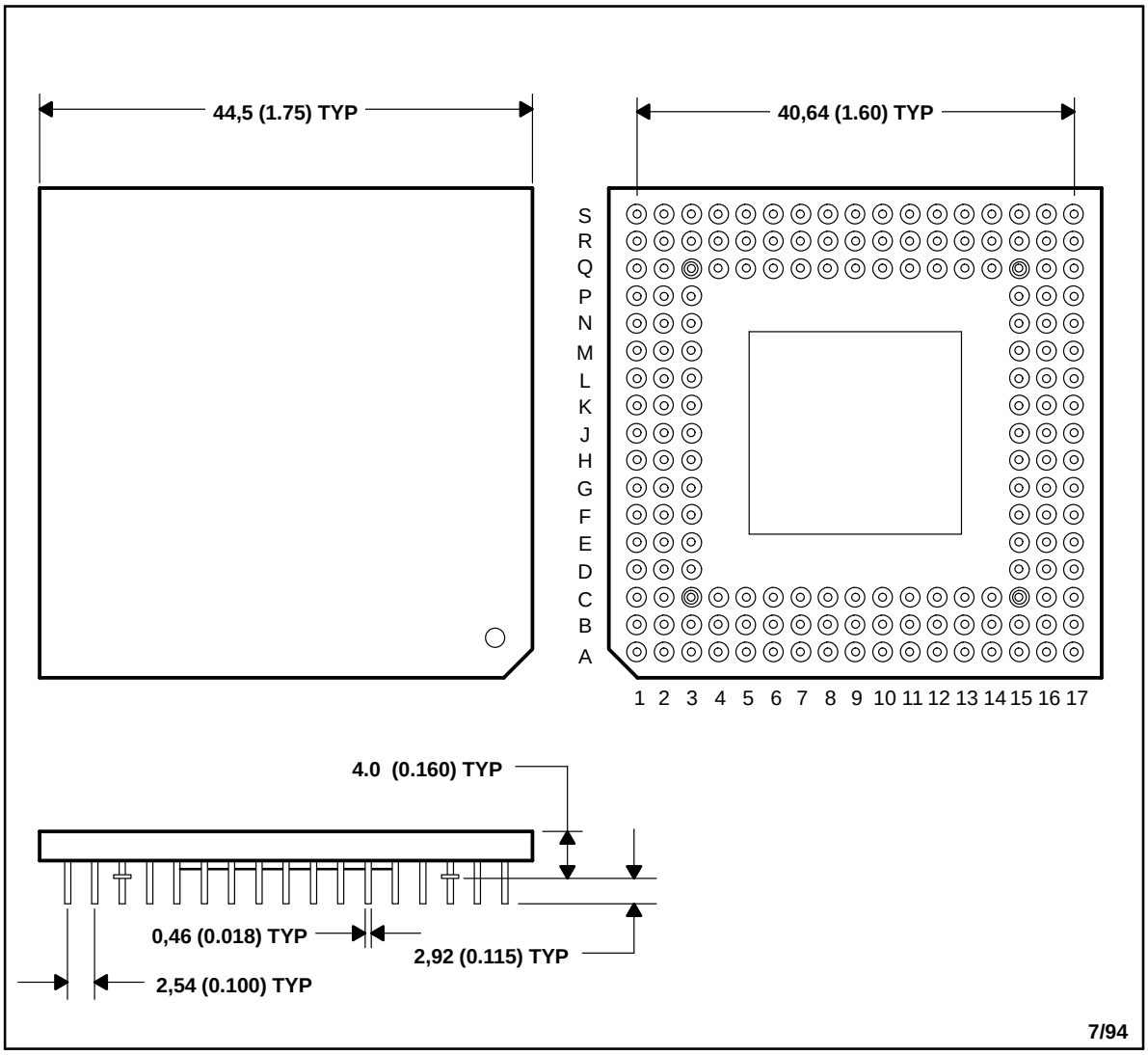
HBN (S-CQFP-G 144) **CERAMIC QUAD FLATPACK**



NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.

Figure 6–11. 168-Pin Ceramic PGA Package Dimensions (TI486SXL)

CPGA-168 PIN **CERAMIC PIN GRID ARRAY**



NOTES: A. All linear dimensions are in millimeters (inches).
B. This drawing is subject to change without notice.

6.3 Thermal Characteristics

The junction-to-ambient (typical) values vary for individual applications. Tha variance depends on the following factors:

- Circuit trace density of the printed circuit board (PCB) and/or the presence or absence of ground or power planes internal to the PCB. These factors affect the conduction of heat away from the device.
- Whether the device is soldered to the PCB or is inserted into a socket
- Orientation of the PCB that the device is mounted on and the proximity of adjacent PCBs or system enclosure. These factors impede the natural convection air circulation around the device.
- Ambient air temperature in close proximity to the device and the proximity of other high-power devices in the system
- Presence of airflow over the device and the attachment of an external heat sink as indicated by the data in Table 6–10 and Table 6–11

For the 100-pin and 144-pin QFPs, the values shown for thermal resistance in Table 6–10 and Table 6–12 with a heatsink are examples of the *estimated* improvement in thermal performance.

Note:

You are responsible for verifying your designs that incorporate any version of a TI microprocessor. Recommended case temperature extremes are specified in Table 5–4, Table 5–5, and Table 5–6.

Table 6–10. TI486SXLC 100-Pin PQFP Thermal Resistance and Airflow

	TI486SXLC 100-Pin PGFP Thermal Resistance (°C/W)		
	Without Heatsink		With Heatsink†
Airflow (Ft/Min)	$R_{\theta JC}$	$R_{\theta JA}$	$R_{\theta JA}$
0	2	36	32
100	2	32	24
200	2	26	18
400	2	19	14
600	2	15	12

† Round, omni-directional heatsink. Dimensions are approximately 1.125 in diameter by 0.42 in high.

Table 6–11. TI486SXL 132-Pin CPGA Thermal Resistance and Airflow

Airflow (Ft/Min)	TI486SXL 132-Pin CPGA† Thermal Resistance (°C/W)	
	$R_{\theta JC}$	$R_{\theta JA}$
0	3	20
100	3	17
200	3	15
400	3	11
600	3	9

† Thermal resistance values shown are based on measurements made on similar ceramic PGA packages.

Table 6–12. TI486SXL 144-Pin PQFP Thermal Resistance and Airflow

Airflow (Ft/Min)	TI486SXL 144-Pin PQFP‡ Thermal Resistance (°C/W)		
	Without Heatsink		With Heatsink§
	$R_{\theta JC}$	$R_{\theta JA}$	$R_{\theta JA}$
0	2	25	18
100	2	21	13
200	2	19	9
400	2	14	7
600	2	12	6

‡ Values shown are based on measurements made on similar 28 mm QFP packages.
§ Pin-Fin heatsink. Dimensions are approximately 1.2” long, by 1.3” wide, by 0.49” high.

Table 6–13. TI486SXL 144-Pin CQFP Thermal Resistance and Airflow

Airflow (Ft/Min)	TI486SXL 144-Pin CQFP¶ Thermal Resistance (°C/W)	
	$R_{\theta JC}$	$R_{\theta JA}$
0	3	33
100	3	28
200	3	24

¶ Thermal resistance values shown are based on measurements made on similar ceramic QFP packages.

Table 6–14. TI486SXL 168-Pin CPGA Thermal Resistance and Airflow

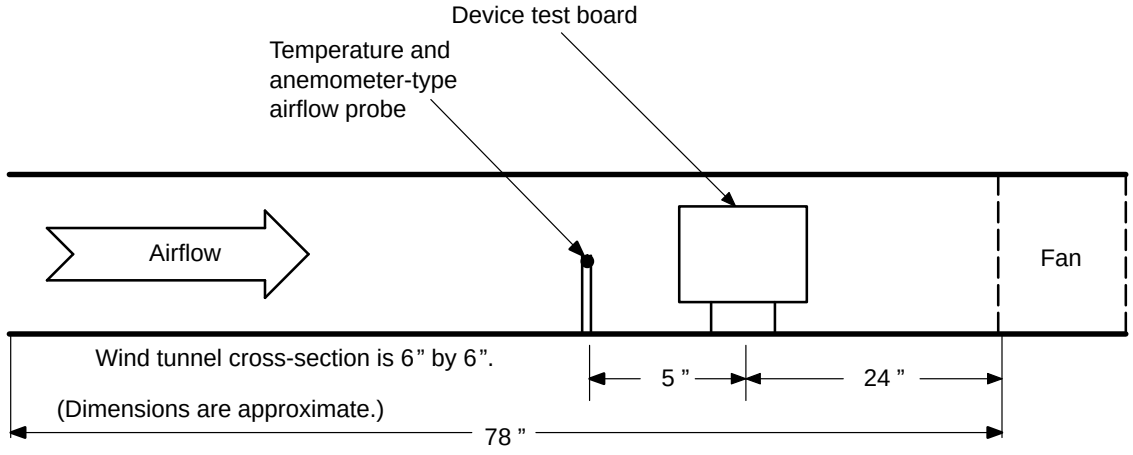
Airflow (Ft/Min)	168-Pin Ceramic PGA Package Thermal Resistance (°C/W) $R_{\theta JC}$	$R_{\theta JA}$
0	3	18
100	3	15
200	3	13
400	3	10
600	3	8

Thermal resistance values shown are based on measurements made on similar ceramic PGA packages.

6.3.1 Airflow Measurement Setup

The wind tunnel used for airflow measurements is represented schematically in Figure 6–12.

Figure 6–12. Wind Tunnel Schematic Diagram



Typically, the devices undergoing thermal test are mounted on a test board consisting of 0.062” thick FR4 printed-circuit-board-material with one-ounce copper etch. Surface-mount devices are soldered to the test board using matching footprints with minimal circuit trace density required to interconnect the device to the board electrically. PGA devices are typically inserted in a socket that is soldered to the test board.

6.3.2 Thermal Parameter Definitions

The maximum die temperature (T_{Jmax}) and the maximum ambient temperature (T_{Amax}) can be calculated using the following equations:

$$T_{Jmax} = T_C + (P_{max} \times R_{\theta JC})$$
$$T_{Amax} = T_J - (P_{max} \times R_{\theta JA)})$$

where:

T_{Jmax} = Maximum average junction temperature (°C)
T_C = Case temperature at top center of package (°C)
P_{max} = Maximum device power dissipation (W)
$R_{\theta JC}$ = Junction-to-case thermal resistance (°C/W)
T_{Amax} = Maximum ambient temperature (°C)
T_J = Average junction temperature (°C)
$R_{\theta JA}$ = Junction-to-ambient thermal resistance (°C/W)

Values for $R_{\theta JA}$ and $R_{\theta JC}$ are given in Table 6–10 and Table 6–11 for various airflows.

Chapter 7

Instruction Set

This chapter provides information pertaining to the TI486SXL(C) microprocessor instruction set. The information explains the general instruction format, fields, flags, clock-count summary, and the instruction encodings. All instructions are listed in the instruction set in Section 7.5, *Instruction Set*.

Topic **Page**

7.1 General Instruction Format

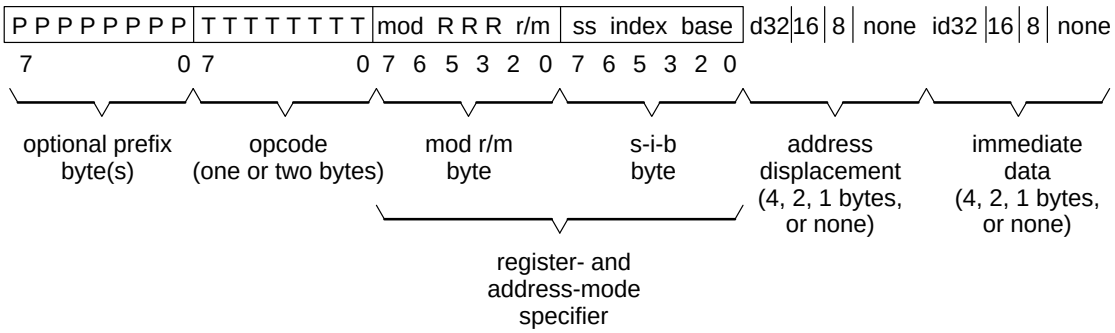
All of the TI486SXL(C) microprocessor family machine instructions follow the general instruction format shown in Figure 7–1. These instructions vary in length and can start at any byte address.

An instruction consists of one or more bytes that can include the following:

- prefix byte(s)
- at least one opcode byte
- a mod r/m byte
- an s-i-b (ss, index, and base fields) byte
- address displacement byte(s)
- immediate data byte(s)

An instruction can be as short as one byte or 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 7–1. General Instruction Format



P – prefix bit
T – opcode bit
R – opcode bit or reg bit

7.2 Instruction Fields

The general instruction format shows the larger fields that make up an instruction. Certain instructions have smaller encoding fields that vary according to the class of operation. These fields define information such as the direction of the operation, the size of the displacements, register encoding, and the sign extension. All the fields are described in Table 7–1. Subsequent subsections provide greater detail.

Table 7–1. Instruction Fields

Field Name	Description	Number of Bits
Prefix	Specifies segment register override, address and operand size, repeat elements in string instruction, and LOCK# assertion	8 per byte
Opcode	Identifies instruction operation	1 or 2 bytes
w	Specifies if data is byte or full size (full size is 16 or 32 bits)	1
d	Specifies direction of data operation	1
s	Specifies if an immediate data field must be sign-extended	1
reg	General register specifier	3
mod r/m	Address mode specifier	2 for mod, 3 for r/m
ss	Scale factor for scaled-index-address mode	2
index	General register to be used as index register	3
base	General register to be used as base register	2
sreg2	Segment register for CS, SS, DS, and ES	2
sreg3	Segment register for CS, SS, DS, ES, FS, and GS	3
eee	Control-, debug-, and test-register specifier	3
Address displacement	Address-displacement operand	1, 2, or 4 bytes
Immediate data	Immediate-data operand	1, 2, or 4 bytes

7.2.1 Prefixes

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

1) Segment override explicitly specifies which segment register an instruction uses.
2) Address size and operand size toggle between 16- and 32-bit addressing modes. Prefixing the instruction for operand size or address size selects the inverse of the current addressing mode. See *also* Section 2.1, *Processor Initialization*, page 2-2.
3) Repeat is used with a string instruction that causes the instruction to be repeated for each element of the string.
4) Lock asserts the hardware LOCK# signal during execution of the instruction.

Table 7–2 lists the encodings for each of the available prefix bytes. The operand-size and address-size prefixes allow individual instructions to override the default value for operand size and effective-address size. The presence of these prefixes selects the opposite (nondefault) operand size and/or effective-address size.

Table 7–2. 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.

7.2.2 Opcode Field

The opcode field is either one or two bytes long and 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 group opcodes use an opcode extension field of three bits in the following byte, called the MOD R/M byte, to resolve the operation type. Opcodes for the entire TI486SXL(C) microprocessor instruction set are listed in Table 7–17 on page 7-14. The opcodes are given in hex values unless shown within brackets ([ ]). Values shown in brackets are binary values.

7.2.3 w Field

The 1-bit field indicates the operand size during 16- and 32-bit data operations as shown in Table 7–3.

Table 7–3. 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

7.2.4 d Field

The d field determines which operand is taken as the source operand and which operand is taken as the destination as shown in Table 7–4.

Table 7–4. d Field Encoding

d Field	Direction Of Operation	Source Operand	Designation Operand
0	Register → Register/Memory	reg	mod r/m or mod ss-index-base
1	Register/Memory → Register	mod r/m or mod ss-index-base	reg

7.2.5 reg Field

The reg field determines which general registers are to be used. The selected register is dependent on whether 16- or 32-bit operation is current and the status of the w bit as shown in Table 7–5.

Table 7–5. reg Field Encoding

reg Field	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

7.2.6 mod and r/m Fields

The mod and r/m sub-fields, which are 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 7–6 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 7–7.

Table 7–6. mod r/m Field Encoding

mod r/m	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	SSS:[BP+SI]	DS:[EDX]
00 011	SS:[BP+DI]	DS:[EBX]
00 100	DS:[SI]	s-i-b is present (see subsection 7.2.7)
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:[EAX+d8]
01 010	SS:[BP+SI+d8]	DS:[EDX+d8]
01 011	SS:[BP+DI+d8]	DS:[EBX+d8]
01 100	DS:[SI+d8]	s-i-b is present (see subsection 7.2.7)
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	SS:[BP+SI+d16]	DS:[EDX+d32]
10 011	SS:[BP+DI+d16]	DS:[EBX+d32]
10 100	DS:[SI+d16]	s-i-b is present (see subsection 7.2.7)
10 101	DS:[DI+d16]	SS:[EBP+d32]
10 110	SS:[BP+d16]	DS:[ESI+d32]
10 111	DS:[BX+d16]	DS:[EDI+d32]
11 000 11 111	See Table 7–7	See Table 7–7

Table 7–7. 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

7.2.7 mod and base Fields

In Table 7–6, the note “s-i-b is present” (for certain entries) forces the use of the mod and base fields as listed in Table 7–8.

Table 7–8. mod base Field Encoding

mod r/m	32-Bit Address Mode With mod r/m Byte and No s-i-b Byte 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:[EBP+(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]

7.2.8 ss Field

The ss field (Table 7–9) 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 7–9. ss Field Encoding

ss Field	Scale Factor
00	x1
01	x2
10	x4
11	x8

7.2.9 index Field

The index field (Table 7–10) specifies the index register used by the offset mechanism for offset-address calculation. When no index register is used (index field = 00), the ss value must be 00 or the effective address is undefined.

Table 7–10. index Field Encoding

index Field	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	none
101	EBP
110	ESI
111	EDI

7.2.10 sreg2 Field

The sreg2 field (Table 7–11) is a two-bit field that allows one of the four 286-type segment registers to be specified.

Table 7–11. sreg2 Field Encoding

sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

7.2.11 sreg3 Field

The sreg3 field (Table 7–12) is a three-bit field that is similar to the sreg2 field, but allows use of the FS and GS segment registers.

Table 7–12. sreg3 Field Encoding

sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	undefined
111	undefined

7.2.12 eee Field

The eee field is used to select the control, debug, and test registers as indicated in Table 7–13. The values shown are the only valid encodings for the eee bits.

Table 7–13. 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

7.3 Flags

The instruction set summary table lists nine flags that instruction execution affects. The conventions shown in Table 7–14 identify the different flags. Table 7–15 lists the conventions used to indicate what action the instruction has on the particular flag.

Table 7–14. 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 7–15. Action of Instruction on Flag

Instruction Table Symbol	Action
m	Flag is modified by the instruction
u	Flag is not changed by the instruction
0	Flag is reset to 0
1	Flag is set to 1

7.4 Clock-Count Summary

The instruction clock-count values presented in Table 7–17 are based on assumptions associated with each individual instruction. Abbreviations indicate the clock-count conditions to simplify the presentation.

7.4.1 Assumptions

The clock-count values assume the following:

- The instruction has been prefetched, decoded, and is ready for execution.
- Bus cycles do not require wait states.
- There are no local-bus HOLD requests delaying processor access to the bus.
- No exceptions are detected during instruction execution.
- 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 1 to the clock count shown.
- All clock counts assume aligned 16-bit memory/IO operands for cache-miss counts.
- If instructions access a misaligned 16-bit operand or a 32-bit operand on even addresses, add two clock counts for read or write, and add four clock counts for read and write.
- If instructions access a 32-bit operand on odd addresses, add four clock counts for read or write, and add eight clock counts for read and write.

7.4.2 Abbreviations

The clock-count values listed in the 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 7–16.

Table 7–16. Clock-Count Abbreviations

Clock-Count Symbol	Explanation
/	Register operand/memory operand
n	Number of times operation is repeated
L	Level of the stack frame
|	Conditional jump taken | conditional jump not taken
\	CPL ≤ IOPL \ CPL > IOPL

7.5 Instruction Set

The TI486SXLC and TI486SXL instruction set is provided in Table 7–17. Instruction name, encoding, flags that are affected, and instruction clock counts for each instruction are shown. The clock-count values are based on the assumptions described in subsection 7.4.1.

Table 7–17. Instruction Set

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
AAA *ASCII Adjust AL after Add*	37	u	u	u	u	u	u	m	u	m	5		5			
AAD *ASCII Adjust AX before Divide*	D5 0A	u	u	u	u	m	m	u	m	u	4		4			
AAM *ASCII Adjust AX after Multiply*	D4 0A	u	u	u	u	m	m	u	m	u	17		17			
AAS *ASCII Adjust AL after Subtract*	3F	u	u	u	u	u	u	m	u	m	5		5			
ADC *Add with Carry*		m	u	u	u	m	m	m	m	m					1	2
Register to Register	1 [00dw] [11 reg r/m]										1		1			
Register to Memory	1 [000w] [mod reg r/m]										3	5	3	5		
Memory to Register	1 [001w] [mod reg r/m]										3	5	3	5		
Immediate to Register/Memory	8 [00sw] [mod 010 r/m]†										1/3	5	1/3	5		
Immediate to Accumulator	1 [010w]†										1		1			
ADD *Integer Add*		m	u	u	u	m	m	m	m	m					1	2
Register to Register	0 [00dw] [11 reg r/m]										1		1			
Register to Memory	0 [000w] [mod reg r/m]										3	5	3	5		
Memory to Register	0 [001w] [mod reg r/m]										3	5	3	5		
Immediate to Register/Memory	8 [00sw] [mod 000 r/m]†										1/3	5	1/3	5		
Immediate to Accumulator	0 [010w]†										1		1			
AND *Boolean AND*		0	u	u	u	m	m	u	m	0					1	2
Register to Register	2 [00dw] [11 reg r/m]										1		1			
Register to Memory	2 [000w] [mod reg r/m]										3	5	3	5		
Memory to Register	2 [001w] [mod reg r/m]										3	5	3	5		
Immediate to Register/Memory	8 [00sw] [mod 100 r/m]†										1/3	5	1/3	5		
Immediate to Accumulator	2 [010w]†										1		1			
ARPL *Adjust Requested Privilege Level*		u	u	u	u	u	m	u	u	u					3	2
From Register/Memory	63 [mod reg r/m]												6/10	10		
BOUND *Check Array Boundaries*	62 [mod reg r/m]	u	u	u	u	u	u	u	u	u					1,4	2,5,6,7,8
If Out of range (Int 5)											11+int		11+int			
If In Range											11		11			
BSF *Scan Bit Forward*		u	u	u	u	u	m	u	u	u					1	2
Register/Memory, Register	0F BC[mod reg r/m]										5/7+n	9+n	5/7+n	9+n		

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
BSR *Scan Bit Reverse*																
Register/Memory, Register	0F BC[mod reg r/m]	u	u	u	u	u	m	u	u	u	5/7+n	9+n	5/7+n	9+n	1	2
BSWAP *Byte Swap*	0F C[1 reg]	u	u	u	u	u	u	u	u	u	5		5			
BT *Test Bit*																
Register/Memory, Immediate	0F BA[mod 100 r/m]†	u	u	u	u	u	u	u	u	m	3/4	5	3/4	5	1	2
Register/Memory, Register	0F A3[mod reg r/m]										3/6	7	3/6	7		
BTC *Test Bit and Complement*																
Register/Memory, Immediate	0F BA[mod 111 r/m]†	u	u	u	u	u	u	u	u	m	4/5	6	4/5	6	1	2
Register/Memory, Register	0F BB[mod reg r/m]										5/8	9	5/8	9		
BTR *Test Bit and Reset*																
Register/Memory, Immediate	0F BA[mod 110 r/m]†	u	u	u	u	u	u	u	u	m	4/5	6	4/5	6	1	2
Register/Memory, Register	0F B3[mod reg r/m]										5/8	9	5/8	9		
BTS *Test Bit and Set*																
Register/Memory	0F BA[mod 101 r/m]	u	u	u	u	u	u	u	u	m	4/5	6	4/5	6	1	2
Register (short form)	0F AB[mod reg r/m]										5/8	9	5/8	9		

† = immediate data ‡ = 8-bit displacement § = 16-bit displacement ¶ = 32-bit displacement m = Flag modified u = Flag unchanged

Notes:
1) Exception 13 fault (general protection) occurs 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) occurs in real mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
2) Exception 13 fault occurs 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.
3) This is a protected mode instruction. Attempted execution in real mode will result in exception 6 (invalid opcode).
4) An exception may occur, depending on the value of the operand.
5) LOCK# is asserted during descriptor table accesses.
6) All segment descriptor accesses in the GDT or LDT made by this instruction automatically asserts LOCK# to maintain descriptor integrity in multiprocessor systems.
7) JMP, CALL, INT, RET, and IRET instructions referring to another code segment causes an exception 13, if an applicable privilege rule is violated.
8) 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 occurs.

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
CALL *Subroutine Call*		u	u	u	u	u	u	u	u	u					1	2,6,7,8
Direct within Segment	E8¶										7		7			
Register/Memory Indirect within Segment	FF [mod 010 r/m]										8/9	10	8/9	10		
Direct Intersegment	9A [unsigned full offset, selector]										12		30			
Call Gate to Same Privilege													41	49		
Call Gate to Different Privilege No Parameters													83	97		
Call Gate to Different Privilege Parameters													81+4x	95+4x		
16-Bit Task to 16-bit TSS													262	263		
16-Bit Task to 32-bit TSS													293	317		
16-Bit Task to V86 Task													179	206		
32-Bit Task to 16-bit TSS													238	258		
32-Bit Task to 32-bit TSS													296	340		
32-Bit Task to V86 Task													182	229		
Indirect Intersegment	FF [mod 011 r/m]										14	17	14	34		
Call Gate to Same Privilege													43	51		
Call Gate to Different Privilege No Parameters													85	99		
Call Gate to Different Privilege Parameters													86+4x	100+4x		
16-Bit Task to 16-bit TSS													267	268		
16-Bit Task to 32-bit TSS													298	322		
16-Bit Task to V86 Task													181	211		
32-Bit Task to 16-bit TSS													243	263		
32-Bit Task to 32-bit TSS													301	345		
32-Bit Task to V86 Task													184	230		
CBW *Convert Byte to Word*	98	u	u	u	u	u	u	u	u	u	3		3			
CDQ *Convert Doubleword to Quadword*	99	u	u	u	u	u	u	u	u	u	1		2			
CLC *Clear Carry Flag*	F8	u	u	u	u	u	u	u	u	0	1		1			
CLD *Clear Direction Flag*	FC	u	0	u	u	u	u	u	u	u	1		1			
CLI *Clear Interrupt Flag*	FA	u	u	0	u	u	u	u	u	u	5		5			9
CLTS *Clear Task Switched Flag*	0F 06	u	u	u	u	u	u	u	u	u	4		4		10	11
CMC *Complement the Carry Flag*	F5	u	u	u	u	u	u	u	u	m	1		1			
CMP *Compare Integers*		m	u	u	u	m	m	m	m	m					1	2
Register to Register	3 [10dw] [11 reg r/m]										1		1			
Register to Memory	3 [101w] [mod reg r/m]										3	5	3	5		
Memory to Register	3 [100w] [mod reg r/m]										3	5	3	5		
Immediate to Register/Memory	8 [00sw] [mod 111 r/m]†										1/3	5	1/3	5		
Immediate to Accumulator	3 [110w]†										1		1			

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
CMPS *Compare String*	A [011w]	m	u	u	u	m	m	m	m	m	8	9	8	9	1	2
CMPXCHG *Compare and Exchange* Register1, Register2 Memory, Register	 0F B[000w] [11 reg2 reg1] 0F B[000w] [mod reg r/m]	m	u	u	u	m	m	m	m	m	 5 7	 8	 5 7	 8		
CWD *Convert Word to Doubleword*	99	u	u	u	u	u	u	u	u	u	1		2			
CWDE *Convert Word to Doubleword Extended*	98	u	u	u	u	u	u	u	u	u	3		3			
DAA *Decimal Adjust AL after Add*	27	u	u	u	u	m	m	m	m	m	4		4			
DAS *Decimal Adjust AL after Subtract*	2F	u	u	u	u	m	m	m	m	m	4		4			
DEC *Decrement by 1* Register/Memory Register (short form)	 F [111w] [mod 001 r/m] 4 [1 reg]	m	u	u	u	m	m	m	m	u	 1/3 1	 5	 1/3 1	 5	1	2
DIV *Unsigned Divide* Accumulator by Register/Memory Divisor: Byte Word Doubleword	F [011w] [mod 110 r/m]	u	u	u	u	u	u	u	u	u	 13/15 21/22 38/39	 17 24 40	 13/15 21/22 38/39	 17 24 40	1,4	2,4
ENTER *Enter New Stack Frame* Level = 0 Level = 1 Level (L) > 1	C8 [8-bit level]§	u	u	u	u	u	u	u	u	u	 7 10 6+4*L	 10 6+4*L	 7 10 6+4*L	 10 6+4*L	1	2
HLT Halt	F4	u	u	u	u	u	u	u	u	u	3		3			11

† = immediate data ‡ = 8-bit displacement § = 16-bit displacement ¶ = 32-bit displacement m = Flag modified u = Flag unchanged

Notes:
1) Exception 13 fault (general protection) occurs 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) occurs in real mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
2) Exception 13 fault occurs 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.
3) This is a protected mode instruction. Attempted execution in real mode will result in exception 6 (invalid opcode).
4) An exception may occur, depending on the value of the operand.
5) LOCK# is asserted during descriptor table accesses.
6) All segment descriptor accesses in the GDT or LDT made by this instruction automatically asserts LOCK# to maintain descriptor integrity in multiprocessor systems.
7) JMP, CALL, INT, RET, and IRET instructions referring to another code segment causes an exception 13, if an applicable privilege rule is violated.
8) 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 occurs.
9) An exception 13 fault occurs if CPL is greater than IOPL.
10) This instruction may be executed in real mode. In real mode, its purpose is primarily to initialize the CPU for protected mode.
11) An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
IDIV *Integer (Signed) Divide*		u	u	u	u	u	u	u	u	u					1,4	2,4
Accumulator by Register/Memory	F [011w] [mod 111 r/m]															
Divisor: Byte											14/15	18	14/15	18		
Word											23/24	25	23/24	25		
Doubleword											40/41	44	40/41	44		
IMUL *Integer (Signed) Multiply*		m	u	u	u	u	u	u	u	m					1	2
Accumulator by Register/Memory	F [011w] [mod 101 r/m]															
Multiplier: Byte											3/5	7	3/5	7		
Word											3/5	7	3/5	7		
Doubleword											7/9	13	7/9	13		
Register with Register/Memory	0F AF[mod reg r/m]															
Multiplier: Byte											3/5	7	3/5	7		
Word											3/5	7	3/5	7		
Doubleword											7/9	13	7/9	13		
Register/Memory with Immediate to Register2	6 [10s1] [mod reg r/m]†															
Multiplier: Byte											3/5	7	3/5	7		
Word											3/5	7	3/5	7		
Doubleword											7/9	13	7/9	13		
IN *Input from I/O Port*		u	u	u	u	u	u	u	u	u						9
Fixed Port	E [010w] [port number]										16	16	16	17		
Variable Port	E [110w]										16	16	16	17		
INC *Increment by 1*		m	u	u	u	m	m	m	m	u					1	2
Register/Memory	F [111w] [mod 000 r/m]										1/3	5	1/3	5		
Register (short from)	4 [0 reg]										1		1			
INS *Input String from I/O Port*	6 [110w]	u	u	u	u	u	u	u	u	u	20	20	14/20	6/21	1	2,9
INT *Software Interrupt*		u	m	0	u	u	u	u	u	u					1,4	5,6,7,8
INT i	CD [i]										14	16				
Protected Mode:																
Interrupt or Trap to Same Privilege													57	58		
Interrupt or Trap to Different Privilege													91	92		
16-Bit Task to 16-bit TSS by Task Gate													265	266		
16-Bit Task to 32-bit TSS by Task Gate													296	320		
16-Bit Task to V86 Task by Task Gate													177	205		
32-Bit Task to 16-bit TSS by Task Gate													241	261		
32-Bit Task to 32-bit TSS by Task Gate													299	343		
32-Bit Task to V86 Task by Task Gate													180	232		
V86 to 16-bit TSS by Task Gate													241	261		
V86 to 32-bit TSS by Task Gate													299	343		
V86 to Privilege 0 by Trap Gate/Int Gate													106	114		

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
INT *Software Interrupt* (Continued)		u	m	0	u	u	u	u	u	u					1,4	5,6,7,8
INT3	CC										14	16				
Protected Mode:																
Interrupt or Trap to Same Privilege													57	58		
Interrupt or Trap to Different Privilege													91	92		
16-Bit Task to 16-bit TSS by Task Gate													265	266		
16-Bit Task to 32-bit TSS by Task Gate													296	320		
16-Bit Task to V86 by Task Gate													177	205		
32-Bit Task to 16-bit TSS by Task Gate													241	261		
32-Bit Task to 32-bit TSS by Task Gate													299	343		
32-Bit Task to V86 by Task Gate													180	232		
V86 to 16-bit TSS by Task Gate													241	261		
V86 to 32-bit TSS by Task Gate													299	343		
V86 to Privilege 0 by Trap Gate/Int Gate													106	114		
INT0	CE	u	u	m	0	u	u	u	u	u						
If 0F == 0											1	1	1	1		
If 0F == 1 (INT4)											15	17				
Protected Mode:																
Interrupt or Trap to Same Privilege													57	58		
Interrupt or Trap to Different Privilege													91	92		
16-Bit Task to 16-bit TSS by Task Gate													265	266		
16-Bit Task to 32-bit TSS by Task Gate													296	320		
16-Bit Task to V86 by Task Gate													177	205		
32-Bit Task to 16-bit TSS by Task Gate													241	261		
32-Bit Task to 32-bit TSS by Task Gate													299	343		
32-Bit Task to V86 by Task Gate													180	232		
V86 to 16-bit TSS by Task Gate													241	261		
V86 to 32-bit TSS by Task Gate													299	343		
V86 to Privilege 0 by Trap Gate/Int Gate													106	114		

† = immediate data ‡ = 8-bit displacement § = 16-bit displacement ¶ = 32-bit displacement m = Flag modified u = Flag unchanged

Notes:
1) Exception 13 fault (general protection) occurs 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) occurs in real mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
2) Exception 13 fault occurs 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.
3) This is a protected mode instruction. Attempted execution in real mode will result in exception 6 (invalid opcode).
4) An exception may occur, depending on the value of the operand.
5) LOCK# is asserted during descriptor table accesses.
6) All segment descriptor accesses in the GDT or LDT made by this instruction automatically asserts LOCK# to maintain descriptor integrity in multiprocessor systems.
7) JMP, CALL, INT, RET, and IRET instructions referring to another code segment causes an exception 13, if an applicable privilege rule is violated.
8) 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 occurs.
9) An exception 13 fault occurs if CPL is greater than IOPL.

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
INVD *Invalidate Cache*	0F 08	u	u	u	u	u	u	u	u	u	7		7			
INVLPG *Invalidate TLB Entry*	0F 01[mod 111 r/m]	u	u	u	u	u	u	u	u	u	5		5			
IRET *Interrupt Return*	CF	m	m	m	m	m	m	m	m	m						2,5,6,7,8
Real Mode											14	14				
Protected Mode													16	17		
Within Task to Same Privilege													35	37		
Within Task to Different Privilege													74	78		
16-Bit Task to 16-bit TSS													259	260		
16-Bit Task to 32-bit TSS													290	314		
16-Bit Task to V86 Task													173	203		
32-Bit Task to 16-bit TSS													235	255		
32-Bit Task to 32-bit TSS													295	339		
32-Bit Task to V86 Task													176	226		
JB/JNAE/JC *Jump on Below/Not Above or Equal/Carry*		u	u	u	u	u	u	u	u	u						8
8-Bit displacement	72‡										4|1		4|1			
Full displacement	0F 82¶										5|2		6|3			
JBE/JNA *Jump on Below or Equal/Not Above*		u	u	u	u	u	u	u	u	u						8
8-Bit displacement	76‡										4|1		4|1			
Full displacement	0F 86¶										5|2		6|3			
JCXZ *Jump on CX Zero*	E3‡	u	u	u	u	u	u	u	u	u	7|3		7|3			8
JE/JZ *Jump on Equal/Zero*		u	u	u	u	u	u	u	u	u						8
8-Bit displacement	74‡										4|1		4|1			
Full displacement	0F 84¶										5|2		6|3			
JECXZ *Jump on ECX Zero*	E3‡	u	u	u	u	u	u	u	u	u	7|3		7|3			8
JL/JNGE *Jump on Less/Not Greater or Equal*		u	u	u	u	u	u	u	u	u						8
8-Bit displacement	7C‡										4|1		4|1			
Full displacement	0F 8C¶										5|2		6|3			

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
JLE/JNG *Jump on Less or Equal/Not Greater*		u	u	u	u	u	u	u	u	u						8
8-Bit displacement	7E‡										4|1		4|1			
Full displacement	0F 8E¶										5|2		6|3			
JMP *Unconditional Jump*		u	u	u	u	u	u	u	u	u					1	2,6,7,8
Short	EB‡										4		4			
Direct within Segment	E9¶										5		6			
Register/Memory Indirect within Segment	FF [mod 100 r/m]										7/8	10	8/9	10		
Direct Intersegment	EA [full offset, selector]										9		27			
Call Gate Same Privilege Level													45	45		
16-Bit Task to 16-bit TSS													265	266		
16-Bit Task to 32-bit TSS													296	320		
16-Bit Task to V86 Task													182	209		
32-Bit Task to 16-bit TSS													241	261		
32-Bit Task to 32-bit TSS													299	343		
32-Bit Task to V86 Task													185	232		
Indirect Intersegment	FF [mod 101 r/m]										13	14	39	39		
Call Gate Same Privilege Level													47	47		
16-Bit Task to 16-bit TSS													270	271		
16-Bit Task to 32-bit TSS													301	325		
16-Bit Task to V86 Task													184	214		
32-Bit Task to 16-bit TSS													246	268		
32-Bit Task to 32-bit TSS													304	348		
32-Bit Task to V86 Task													187	237		
JNB/JAE/JNC *Jump on Not Below/ Above or Equal/Not Carry*		u	u	u	u	u	u	u	u	u						8
8-Bit displacement	73‡										4|1		4|1			
Full displacement	0F 83¶										5|2		6|3			

† = immediate data ‡ = 8-bit displacement § = 16-bit displacement ¶ = 32-bit displacement m = Flag modified u = Flag unchanged

Notes:
1) Exception 13 fault (general protection) occurs 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) occurs in real mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
2) Exception 13 fault occurs 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.
3) This is a protected mode instruction. Attempted execution in real mode will result in exception 6 (invalid opcode).
4) An exception may occur, depending on the value of the operand.
5) LOCK# is asserted during descriptor table accesses.
6) All segment descriptor accesses in the GDT or LDT made by this instruction automatically asserts LOCK# to maintain descriptor integrity in multiprocessor systems.
7) JMP, CALL, INT, RET, and IRET instructions referring to another code segment causes an exception 13, if an applicable privilege rule is violated.
8) 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 occurs.

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
JNBE/JA *Jump on Not Below or Equal/Above* 8-Bit displacement Full displacement	 77‡ 0F 87¶	u	u	u	u	u	u	u	u	u	 4|1 5|2		 4|1 6|3			8
JNE/JNZ *Jump on Not Equal/Not Zero* 8-Bit Displacement Full Displacement	 75‡ 0F 85¶	u	u	u	u	u	u	u	u	u	 4|1 5|2		 4|1 6|3			8
JNL/JGE *Jump on Not Less/Greater or Equal* 8-Bit displacement Full displacement	 7D‡ 0F 8D¶	u	u	u	u	u	u	u	u	u	 4|1 5|2		 4|1 6|3			8
JNLE/JG *Jump on Not Less or Equal/Greater* 8-Bit displacement Full displacement	 7F‡ 0F 8F¶	u	u	u	u	u	u	u	u	u	 4|1 5|2		 4|1 6|3			8
JNO *Jump on Not Overflow* 8-Bit displacement Full displacement	 71‡ 0F 81¶	u	u	u	u	u	u	u	u	u	 4|1 5|2		 4|1 6|3			8
JNP/JPO *Jump on Not Parity/Parity Odd* 8-Bit displacement Full displacement	 7B‡ 0F 8B¶	u	u	u	u	u	u	u	u	u	 4|1 5|2		 4|1 6|3			8
JNS *Jump on Not Sign* 8-Bit displacement Full displacement	 79‡ 0F 89¶	u	u	u	u	u	u	u	u	u	 4|1 5|2		 4|1 6|3			8
JO *Jump on Overflow* 8-Bit displacement Full displacement	 70‡ 0F 80¶	u	u	u	u	u	u	u	u	u	 4|1 5|2		 4|1 6|3			8
JP/JPE *Jump on Parity/Parity Even* 8-Bit displacement Full displacement	 7A‡ 0F 8A¶	u	u	u	u	u	u	u	u	u	 4|1 5|2		 4|1 6|3			8
JS *Jump on Sign* 8-Bit displacement Full displacement	 78‡ 0F 88¶	u	u	u	u	u	u	u	u	u	 4|1 5|2		 4|1 6|3			8
LAHF *Load AH with Flags*	9F	u	u	u	u	u	u	u	u	u	2		2			

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
LAR *Load Access Rights* From Register/Memory	0F 02[mod reg r/m]	u	u	u	u	u	m	u	u	u			11/12	14	3	2,5,6,12
LDS *Load Pointer to DS*	C5 [mod reg r/m]	u	u	u	u	u	u	u	u	u	6	7	19	22	1	2,6,13
LEA *Load Effective Address* No Index Register With Index Register	8D [mod reg r/m]	u	u	u	u	u	u	u	u	u	 2 3		 2 3			
LEAVE *Leave Current Stack Frame*	C9	u	u	u	u	u	u	u	u	u	5	6	5	6	1	2
LES *Load Pointer to ES*	C4 [mod reg r/m]	u	u	u	u	u	u	u	u	u	7	8	20	21	1	2,6,13
LFS *Load Pointer to FS*	0F B4[mod reg r/m]	u	u	u	u	u	u	u	u	u	7	8	20	21	1	2,6,13
LGDT *Load GDT Register*	0F 01[mod 010 r/m]	u	u	u	u	u	u	u	u	u	9	9	9	9	1,10	2,11
LGS *Load Pointer to GS*	0F B5[mod reg r/m]	u	u	u	u	u	u	u	u	u	7	8	7	8	1	2,6,13
LIDT *Load IDT Register*	0F 01[mod 011 r/m]	u	u	u	u	u	u	u	u	u	11	11	11	11	1,10	2,11
LLDT *Load LDT Register* From Register/Memory	0F 00[mod 010 r/m]	u	u	u	u	u	u	u	u	u			16/17	18	3	2,5,6,11
LMSW *Load Machine Status Word* From Register/Memory	0F 01[mod 110 r/m]	u	u	u	u	u	u	u	u	u	5	8	5	8	1,10	2,11
LODS *Load String*	A [110w]	u	u	u	u	u	u	u	u	u	6	6	6	6	1	2
LOOP *Offset Loop/No Loop*	E2‡	u	u	u	u	u	u	u	u	u	8|4		9|4			8

† = immediate data ‡ = 8-bit displacement § = 16-bit displacement ¶ = 32-bit displacement m = Flag modified u = Flag unchanged

Notes:
1) Exception 13 fault (general protection) occurs 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) occurs in real mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
2) Exception 13 fault occurs 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.
3) This is a protected mode instruction. Attempted execution in real mode will result in exception 6 (invalid opcode).
4) An exception may occur, depending on the value of the operand.
5) LOCK# is asserted during descriptor table accesses.
6) All segment descriptor accesses in the GDT or LDT made by this instruction automatically asserts LOCK# to maintain descriptor integrity in multiprocessor systems.
7) JMP, CALL, INT, RET, and IRET instructions referring to another code segment causes an exception 13, if an applicable privilege rule is violated.
8) 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 occurs.
9) An exception 13 fault occurs if CPL is greater than IOPL.
10) This instruction may be executed in real mode. In real mode, its purpose is primarily to initialize the CPU for protected mode.
11) An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).
12) Any violation of privilege rules applied to the selector operand does not cause a protection exception. Rather, the zero flag is cleared.
13) For segment load operations, the CPL, RPL, and DPL must agree with the privolege rules to avoid an exception 13 fault. The segment's descriptor must indicate present or exception 11 occurs (DS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 occurs.

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
LOOPNZ/LOOPNE *Offset*	E0‡	u	u	u	u	u	u	u	u	u	8|4		9|4			8
LOOPZ/LOOPE *Offset*	E1‡	u	u	u	u	u	u	u	u	u	8|4		9|4			8
LSL *Load Segment Limit*		u	u	u	u	u	m	u	u	u					3	2,5,6,12
From Register/Memory	0F 03[mod reg r/m]												14/15	17		
LSS *Load Pointer to SS*	0F B2[mod reg r/m]	u	u	u	u	u	u	u	u	u	7	8	19	20	3	2,6,13
LTR *Load Task Register*		u	u	u	u	u	u	u	u	u					3	2,5,6,11
From Register/Memory	0F 00[mod reg r/m]												16/17	18		
MOV *Move Data*		u	u	u	u	u	u	u	u	u					1	2,6,13
Register to Register/Memory	8 [110w] [mod reg r/m]										1/2	2	1/2	2		
Register/Memory to Register	8 [101w] [mod reg r/m]										1/2	4	1/2	4		
Immediate to Register/Memory	C [011w] [mod 000 r/m]†										1/2	2	1/2	2		
Immediate to Register (short form)	B [w reg]†										1		1			
Memory to Accumulator (short form)	A [000w]¶										2	4	2	4		
Accumulator to Memory (short form)	A [001w]¶										2	2	2	2		
Register/Memory to Segment Register	8E [mod sreg3 r/m]										2/3	5	15/16	18		
Segment Register to Register/Memory	8C [mod reg r/m]										1/3	3	1/3	3		
MOV *Move to/from Control/Debug/Test Registers*		u	u	u	u	u	u	u	u	u						11
Register to CR0/CR2/CR3	0F 22[11 eee reg]										14/3/3		14/3/3			
CR0/CR2/CR3 to Register	0F 20[11 eee reg]										2/3/3		2/3/3			
Register to DR0–DR3	0F 23[11 eee reg]										10		10			
DR0–DR3 to Register	0F 21[11 eee reg]										9		9			
Register to DR6–DR7	0F 23[11 eee reg]										10		10			
DR6–DR7 to Register	0F 21[11 eee reg]										9		9			
Register to TR3–5	0F 26[11 eee reg]										10		10			
TR3–5 to Register	0F 24[11 eee reg]										11		11			
Register to TR6–TR7	0F 26[11 eee reg]										8		8			
TR6–TR7 to Register	0F 24[11 eee reg]										9		9			
MOVS *Move String*	A [010w]	u	u	u	u	u	u	u	u	u	5	5	5	5	1	2
MOVSX *Move with Sign Extension*		u	u	u	u	u	u	u	u	u					1	2
Register from Register/Memory	0F B[111w] [mod reg r/m]										2/3	5	2/3	5		
MOVZX *Move with Zero Extension*		u	u	u	u	u	u	u	u	u					1	2
Register from Register/Memory	0F B[011w] [mod reg r/m]										2/3	5	2/3	5		

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
MUL *Unsigned Multiply*	F [011w] [mod 100 r/m]	m	u	u	u	u	u	u	u	m					1	2
Accumulator with Register/Memory																
Multiplier: Byte											3/5	7	3/5	7		
Word											3/5	7	3/5	7		
Doubleword											10/9	14	10/9	14		
NEG *Negate Integer*	F [011w] [mod 011 r/m]	m	u	u	u	m	m	m	m	m	1/3	5	1/3	5	1	2
NOP *No Operation*	90	u	u	u	u	u	u	u	u	u	1		1			
NOT *Boolean Complement*	F [011w] [mod 010 r/m]	u	u	u	u	u	u	u	u	u	1/3	5	1/3	5	1	2
OR *Boolean OR*		0	u	u	u	m	m	m	m	0					1	2
Register to Register	0 [10dw] [11 reg r/m]										1		1			
Register to Memory	0 [100w] [mod reg r/m]										3	5	3	5		
Memory to Register	0 [101w] [mod reg r/m]										3	5	3	5		
Immediate to Register/Memory	8 [000w] [mod 001 r/m]†										1/3	5	1/3	5		
Immediate to Accumulator	0 [110w]†										1		1			
OUT *Output to Port*		u	u	u	u	u	u	u	u	u						9
Fixed Port	E [011w] [port number]										18	18	14\34	14\35		
Variable Port	E [111w]										18	18	14\34	14\35		
OUTS *Output String*	6 [111w]	u	u	u	u	u	u	u	u	u	20	20	14\34	14\34	1	2,9

† = immediate data ‡ = 8-bit displacement § = 16-bit displacement ¶ = 32-bit displacement m = Flag modified u = Flag unchanged

Notes:
1) Exception 13 fault (general protection) occurs 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) occurs in real mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
2) Exception 13 fault occurs 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.
3) This is a protected mode instruction. Attempted execution in real mode will result in exception 6 (invalid opcode).
4) An exception may occur, depending on the value of the operand.
5) LOCK# is asserted during descriptor table accesses.
6) All segment descriptor accesses in the GDT or LDT made by this instruction automatically asserts LOCK# to maintain descriptor integrity in multiprocessor systems.
7) JMP, CALL, INT, RET, and IRET instructions referring to another code segment causes an exception 13, if an applicable privilege rule is violated.
8) 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 occurs.
9) An exception 13 fault occurs if CPL is greater than IOPL.
10) This instruction may be executed in real mode. In real mode, its purpose is primarily to initialize the CPU for protected mode.
11) An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).
12) Any violation of privilege rules applied to the selector operand does not cause a protection exception. Rather, the zero flag is cleared.
13) For segment load operations, the CPL, RPL, and DPL must agree with the privolege rules to avoid an exception 13 fault. The segment's descriptor must indicate present or exception 11 occurs (DS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 occurs.

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
POP *Pop Value off Stack*		u	u	u	u	u	u	u	u	u					1	2,6,13
Register/Memory	8F [mod 000 r/m]										3/5	4/5	3/5	4/5		
Register (short form)	5 [1 reg]										3	4	3	4		
Segment Register (ES, CS, SS, DS)	[000 sreg2 110]										8	9	8	9		
Segment Register (ES, CS, SS, DS, FS, GS)	0F [10 sreg3 001]										8	9	8	9		
POPA *Pop All General Registers*	61	u	u	u	u	u	u	u	u	u	18	18	18	18	1	2
POPF *Pop Stack into Flags*	9D	m	m	m	m	m	m	m	m	m	4	5	4	5	1	2,14
PREFIX *BYTES*		u	u	u	u	u	u	u	u	u						9
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*		u	u	u	u	u	u	u	u	u					1	2
Register/Memory	FF [mod 110 r/m]										2/4	4	2/4	4		
Register (short form)	5 [0 reg]										2	2	2	2		
Segment Register (ES, CS, SS, DS)	[000 sreg2 110]										2	2	2	2		
Segment Register (ES, CS, SS, DS, FS, GS)	0F [10 sreg3 000]										2	2	2	2		
Immediate	6 [10s0]†										2	2	2	2		
PUSHA *Push All General Registers*	60	u	u	u	u	u	u	u	u	u	17	17	17	17	1	2
PUSHF *Push Flags Register*	9C	u	u	u	u	u	u	u	u	u	2	2	2	2	1	2
RCL *Rotate Through Carry Left*		m	u	u	u	u	u	u	u	m					1	2
Register/Memory by 1	D [000w] [mod 010 r/m]										9/9	10	9/9	10		
Register/Memory by CL	D [001w] [mod 010 r/m]										9/9	10	9/9	10		
Register/Memory by Immediate	C [000w] [mod 010 r/m]†										9/9	10	9/9	10		
RCR *Rotate Through Carry Right*		m	u	u	u	u	u	u	u	m					1	2
Register/Memory by 1	D [000w] [mod 011 r/m]										9/9	10	9/9	10		
Register/Memory by CL	D [001w] [mod 011 r/m]										9/9	10	9/9	10		
Register/Memory by Immediate	C [000w] [mod 011 r/m]†										9/9	10	9/9	10		

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
REP INS *Input String*	F2 6[110w]	u	u	u	u	u	u	u	u	u	20+9n	20+9n	5+9n\ 18+9n	5+9n\ 19+9n	1	2,9
REP LODS *Load String*	F2 A[110w]	u	u	u	u	u	u	u	u	u	4+5n	4+5n	4+5n	4+5n	1	2
REP MOVS *Move String*	F2 A[010w]	u	u	u	u	u	u	u	u	u	5+4n	5+4n	5+4n	5+4n	1	2
REP OUTS *Output String*	F2 6[111w]	u	u	u	u	u	u	u	u	u	20+4n	20+4n	5+4n\ 18+4n	5+4n\ 19+4n	1	2,9
REP STOS *Store String*	F2 A[101w]	u	u	u	u	u	u	u	u	u	3+4n	3+4n	3+4n	3+4n	1	2
REPE CMPS *Compare String* (Find nonmatch)	F3 A[011w]	m	u	u	u	m	m	m	m	m	5+8n	5+8n	5+8n	5+8n	1	2
REPE SCAS *Scan String* (Find non-AL/AX/EAX)	F3 A[111w]	m	u	u	u	m	m	m	m	m	4+5n	4+6n	4+5n	4+6n	1	2
REPNE CMPS *Compare String* (Find match)	F2 A[011w]	m	u	u	u	m	m	m	m	m	5+8n	5+8n	5+8n	5+8n	1	2
REPNE SCAS *Scan String* (Find AL/AX/EAX)	F2 A[111w]	m	u	u	u	m	m	m	m	m	4+5n	4+6n	4+5n	4+6n	1	2

† = immediate data ‡ = 8-bit displacement § = 16-bit displacement ¶ = 32-bit displacement m = Flag modified u = Flag unchanged

Notes:
1) Exception 13 fault (general protection) occurs 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) occurs in real mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
2) Exception 13 fault occurs 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.
3) This is a protected mode instruction. Attempted execution in real mode will result in exception 6 (invalid opcode).
4) An exception may occur, depending on the value of the operand.
5) LOCK# is asserted during descriptor table accesses.
6) All segment descriptor accesses in the GDT or LDT made by this instruction automatically asserts LOCK# to maintain descriptor integrity in multiprocessor systems.
7) JMP, CALL, INT, RET, and IRET instructions referring to another code segment causes an exception 13, if an applicable privilege rule is violated.
8) 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 occurs.
9) An exception 13 fault occurs if CPL is greater than IOPL.
10) This instruction may be executed in real mode. In real mode, its purpose is primarily to initialize the CPU for protected mode.
11) An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).
12) Any violation of privilege rules applied to the selector operand does not cause a protection exception. Rather, the zero flag is cleared.
13) For segment load operations, the CPL, RPL, and DPL must agree with the privolege rules to avoid an exception 13 fault. The segment's descriptor must indicate present or exception 11 occurs (DS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 occurs.
14) 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.

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
RET *Return from Subroutine*		u	u	u	u	u	u	u	u	u					1	2,5,6,7,8
Within Segment	C3										10		10			
Within Segment Add Immediate to SP	C2§										10		10			
Intersegment	CB										13	13	26	26		
Intersegment Add Immediate to SP	CA§										13	13	26	27		
Protected Mode: Different Privilege Level																
Intersegment													69	72		
Intersegment Add Immediate to SP													69	72		
ROL *Rotate Left*		m	u	u	u	u	u	u	u	m					1	2
Register/Memory by 1	D [000w] [mod 000 r/m]										2/4	6	2/4	6		
Register/Memory by CL	D [001w] [mod 000 r/m]										3/5	7	3/5	7		
Register/Memory by Immediate	C [000w] [mod 000 r/m]†										2/4	6	2/4	6		
ROR *Rotate Right*		m	u	u	u	u	u	u	u	m					1	2
Register/Memory by 1	D [000w] [mod 001 r/m]										2/4	6	2/4	6		
Register/Memory by CL	D [001w] [mod 001 r/m]										3/5	7	3/5	7		
Register/Memory by Immediate	C [000w] [mod 001 r/m]†										2/4	6	2/4	6		
RSDC *Restore Segment Register and Descriptor*	0F 79 [mod sreg3 r/m]	u	u	u	u	u	u	u	u	u	14		14		15	15
RSLDT *Restore LDTR and Descriptor*	0F 78 [mod 000 r/m]	u	u	u	u	u	u	u	u	u	14		14		15	15
RSM *Resume from SMM Mode*	oF AA	u	u	u	u	u	u	u	u	u	76		76		15	15
RSTS *Restore TSR and Descriptor*	0F 7D [mod 000 r/m]	u	u	u	u	u	u	u	u	u	14		14		15	15
SAHF *Store AH in Flags*	9E	u	u	u	u	m	m	u	m	m	2		2			
SAL *Shift Left Arithmetic*		m	u	u	u	m	m	u	m	m						
Register/Memory by 1	D [000w] [mod 100 r/m]										2/4	6	2/4	6		
Register/Memory by CL	D [001w] [mod 100 r/m]										3/5	7	3/5	7		
Register/Memory by Immediate	C [000w] [mod 100 r/m]†										2/4	6	2/4	6		
SAR *Shift Right Arithmetic*		m	u	u	u	m	m	m	m	m						
Register/Memory by 1	D [000w] [mod 111 r/m]										2/4	6	2/4	6		
Register/Memory by CL	D [001w] [mod 111 r/m]										3/5	7	3/5	7		
Register/Memory by Immediate	C [000w] [mod 111 r/m]†										2/4	5	2/4	8		

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
SBB *Integer Subtract with Borrow*		m	u	u	u	m	m	m	m	m					1	2
Register to Register	1 [10dw] [11 reg r/m]										1		1			
Register to Memory	1 [100w] [mod reg r/m]										3	5	3	5		
Memory to Register	1 [101w] [mod reg r/m]										3	5	3	5		
Immediate to Register/Memory	8 [00sw] [mod 011 r/m]†										1/3	5	1/3	5		
Immediate to Accumulator (short form)	1 [110w]†										1		1			
SCAS *Scan String*	A [111w]	m	u	u	u	m	m	m	m	m	6	6	6	6	1	2
SETB/SETNAE/SETC *Set Byte on Below/ Not Above or Equal/Carry*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 92[mod 000 r/m]										2/2	2	2/2	2		
SETBE/SETNA *Set Byte on Below or Equal/ Not Above*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 96 [mod 000 r/m]										2/2	2	2/2	2		
SETE/SETZ *Set Byte on Equal/Zero Register/ Memory*		u	u	u	u	u	u	u	u	u						2
	0F 94 [mod 000 r/m]										2/2	2	2/2	2		
SETL/SETNGE *Set Byte on Less/ Not Greater or Equal*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 9C[mod 000 r/m]										2/2	2	2/2	2		
SETLE/SETNG *Set Byte on Less or Equal/ Not Greater*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 9E[mod 000 r/m]										2/2	2	2/2	2		

† = immediate data ‡ = 8-bit displacement § = 16-bit displacement ¶ = 32-bit displacement m = Flag modified u = Flag unchanged

Notes:
1) Exception 13 fault (general protection) occurs 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) occurs in real mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
2) Exception 13 fault occurs 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.
3) This is a protected mode instruction. Attempted execution in real mode will result in exception 6 (invalid opcode).
4) An exception may occur, depending on the value of the operand.
5) LOCK# is asserted during descriptor table accesses.
6) All segment descriptor accesses in the GDT or LDT made by this instruction automatically asserts LOCK# to maintain descriptor integrity in multiprocessor systems.
7) JMP, CALL, INT, RET, and IRET instructions referring to another code segment causes an exception 13, if an applicable privilege rule is violated.
8) 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 occurs.
15) All memory accesses using this instruction are noncacheable as this instruction uses SMM address space.

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
SETNB/SETAE/SETNC *Set Byte on Not Below/ Above or Equal/Not Carry*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 93[mod 000 r/m]										2/2	2	2/2	2		
SETNBE/SETA *Set Byte on Not Below or Equal/ Above*		u	u	u	u	u	u	u	u	u						2
To Register Memory	0F 97[mod 000 r/m]										2/2	2	2/2	2		
SETNE/SETNZ *Set Byte on Not Equal/ Not Zero*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 95[mod 000 r/m]										2/2	2	2/2	2		
SETNL/SETGE *Set Byte on Not Less/ Greater or Equal*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 9D [mod 000 r/m]										2/2	2	2/2	2		
SETNLE/SETG *Set Byte on Not Less or Equal/Greater*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 9F[mod 000 r/m]										2/2	2	2/2	2		
SETNO *Set Byte on Not Overflow*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 91[mod 000 r/m]										2/2	2	2/2	2		
SETNP/SETPO *Set Byte on Not Parity/ Parity Odd*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 9B[mod 000 r/m]										2/2	2	2/2	2		
SETNS *Set Byte on Not Sign*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 99[mod 000 r/m]										2/2	2	2/2	2		
SETO *Set Byte on Overflow*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 90[mod 000 r/m]										2/2	2	2/2	2		
SETP/SETPE *Set Byte on Parity/Parity Even*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 9A[mod 000 r/m]										2/2	2	2/2	2		
SETS *Set Byte on Sign*		u	u	u	u	u	u	u	u	u						2
To Register/Memory	0F 98[mod 000 r/m]										2/2	2	2/2	2		
SGDT *Store GDT Register*		u	u	u	u	u	u	u	u	u					1,10	2
To Register/Memory	0F 01[mod 00 r/m]										6	6	6	6		
SHL *Shift Left Logical*		m	u	u	u	m	m	u	m	m					1	2
Register/Memory by 1	D [000w] [mod 100 r/m]										1/3	5	1/3	5		
Register/Memory by CL	D [001w] [mod 100 r/m]										2/4	6	2/4	6		
Register/memory by Immediate	C [000w] [mod 100 r/m]†										1/3	5	1/3	5		

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
SHLD *Shift Left Double*		u	u	u	u	m	m	u	m	m						
Register/memory by Immediate	0F A4[mod reg r/m]†										1/3	5	1/3	5		
Register/Memory by CL	0F A5[mod reg r/m]										3/5	7	3/5	7		
SHR *Shift Right Logical*		m	u	u	u	m	m	u	m	m					1	2
Register/Memory by 1	D [000w] [mod 101 r/m]										1/3	5	1/3	5		
Register/Memory by CL	D [001w] [mod 101 r/m]										2/4	6	2/4	6		
Register/Memory by Immediate	C [000w] [mod 101 r/m]†										1/3	4	1/3	4		
SHRD *Shift Right Double*		u	u	u	u	m	m	u	m	m						
Register/Memory by Immediate	0F AC[mod reg r/m]†										1/3	5	1/3	5		
Register/Memory by CL	0F AD[mod reg r/m]										3/5	7	3/5	7		
SIDT *Store IDT Register*		u	u	u	u	u	u	u	u	u					1,10	2
To Register/Memory	0F 01[mod 001 r/m]										8	8	8	8		
SLDT *Store LDT Register*		u	u	u	u	u	u	u	u	u					3	2
To Register/Memory	0F 00[mod 000 r/m]												2/3	3		
SMSW *Store Machine Status Word*	0F 01[mod 100 r/m]	u	u	u	u	u	u	u	u	u	2/4	4	2/4	4	1,10	2,11
STC *Set Carry Flag*	F9	u	u	u	u	u	u	u	u	1	1		1			
STD *Set Direction Flag*	FD	u	1	u	u	u	u	u	u	u	2		2			
STI *Set Interrupt Flag*	FB	u	u	1	u	u	u	u	u	u	4		4			9

† = immediate data ‡ = 8-bit displacement § = 16-bit displacement ¶ = 32-bit displacement m = Flag modified u = Flag unchanged

Notes:
1) Exception 13 fault (general protection) occurs 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) occurs in real mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
2) Exception 13 fault occurs 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.
3) This is a protected mode instruction. Attempted execution in real mode will result in exception 6 (invalid opcode).
4) An exception may occur, depending on the value of the operand.
5) LOCK# is asserted during descriptor table accesses.
6) All segment descriptor accesses in the GDT or LDT made by this instruction automatically asserts LOCK# to maintain descriptor integrity in multiprocessor systems.
7) JMP, CALL, INT, RET, and IRET instructions referring to another code segment causes an exception 13, if an applicable privilege rule is violated.
8) 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 occurs.
9) An exception 13 fault occurs if CPL is greater than IOPL.
10) This instruction may be executed in real mode. In real mode, its purpose is primarily to initialize the CPU for protected mode.
11) An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
STOS *Store String*	A [101w]	u	u	u	u	u	u	u	u	u	3	3	3	3	1	2
STR *Store Task Register* To Register/Memory	0F 00[mod 001 r/m]	u	u	u	u	u	u	u	u	u			1/2	2	3	2
SUB *Integer Subtract* Register to Register Register to memory Memory to Register Immediate to Register/Memory Immediate to Accumulator (short form)	 2 [10dw] [11 reg r/m] 2 [100w] [mod reg r/m] 2 [101w] [mod reg r/m] 8 [00sw] [mod 101 r/m]† 2 [110w]†	m	u	u	u	m	m	m	m	m	 1 3 3 1/3 1	 5 5 5	 1 3 3 1/3 1	 5 5 5	1	2
SVDC *Save Segment Register and Descriptor*	0F 78 [mod sreg3 r/m]	u	u	u	u	u	u	u	u	u		22		22	15	15
SVLDT *Save LDTR and Descriptor*	0F 7A [mod 000 r/m]	u	u	u	u	u	u	u	u	u		22		22	15	15
SVTS *Save TSR and Descriptor*	0F 7C [mod 000 r/m]	u	u	u	u	u	u	u	u	u		22		22	15	15
TEST *Test Bits* Register/Memory and Register Immediate Data and Register/Memory Immediate Data and Accumulator	 8 [010w] [mod reg r/m] F [011w] [mod 000 r/m]† A [100w]†	0	u	u	u	m	m	u	m	0	 1/3 1/3 1	 5 5	 1/3 1/3 1	 5 5	1	2
VERR *Verify Read Access* To Register/Memory	0F 00[mod 100 r/m]	u	u	u	u	u	m	u	u	u			9/10	12	3	2,5,6,12
VERW *Verify Write Access* To Register/Memory	0F 00[mod 101 r/m]	u	u	u	u	u	m	u	u	u			9/10	12	3	2,5,6,12
WAIT *Wait Until FPU Not Busy*	9B	u	u	u	u	u	u	u	u	u	5	5	5	5		
WBINVD *Write-Back and Invalidate Cache*	0F 09	u	u	u	u	u	u	u	u	u	8		8			
XADD *Exchange and Add* Register1, Register2 Memory, Register	 0FC[000w] [11 reg2 reg1] 0FC[000w] [mod reg r/m]	m	u	u	u	m	m	m	m	m	 3 6	 6	 3 6	 6		
XCHG *Exchange* Register/Memory with Register Register with Accumulator	 8 [011w] [mod reg r/m] 9 [0 reg]	u	u	u	u	u	u	u	u	u	 3/5 3	 5	 3/5 3	 5	1,16	2,16

Table 7–17. Instruction Set (Continued)

Instruction	Opcode	Flags									Real-Mode Clocks		Protected-Mode Clocks		Notes	
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg /Cache Hit	Cache Miss	Reg /Cache Hit	Cache Miss	Real Mode	Protected Mode
XLAT *Translate Byte*	D7	u	u	u	u	u	u	u	u	u	3	5	3	5		2
XOR *Boolean Exclusive OR*		0	u	u	u	m	m	u	m	0					1	2
Register to Register	3 [00dw] [11 reg r/m]										1		1			
Register to Memory	3 [000w] [mod reg r/m]										3	5	3	5		
Memory to Register	3 [001w] [mod reg r/m]										3	5	3	5		
Immediate to Register/Memory	8 [00sw] [mod 110 r/m]†										1/3	5	1/3	5		
Immediate to Accumulator (short form)	3 [010w]†										1		1			

† = immediate data ‡ = 8-bit displacement § = 16-bit displacement ¶ = 32-bit displacement m = Flag modified u = Flag unchanged

Notes:
1) Exception 13 fault (general protection) occurs 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) occurs in real mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
2) Exception 13 fault occurs 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.
3) This is a protected mode instruction. Attempted execution in real mode will result in exception 6 (invalid opcode).
4) An exception may occur, depending on the value of the operand.
5) LOCK# is asserted during descriptor table accesses.
6) All segment descriptor accesses in the GDT or LDT made by this instruction automatically asserts LOCK# to maintain descriptor integrity in multiprocessor systems.
7) JMP, CALL, INT, RET, and IRET instructions referring to another code segment causes an exception 13, if an applicable privilege rule is violated.
8) 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 occurs.
9) An exception 13 fault occurs if CPL is greater than IOPL.
10) This instruction may be executed in real mode. In real mode, its purpose is primarily to initialize the CPU for protected mode.
11) An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).
12) Any violation of privilege rules applied to the selector operand does not cause a protection exception. Rather, the zero flag is cleared.
13) For segment load operations, the CPL, RPL, and DPL must agree with the privolege rules to avoid an exception 13 fault. The segment's descriptor must indicate present or exception 11 occurs (DS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 occurs.
14) 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.
15) All memory accesses using this instruction are noncacheable as this instruction uses SMM address space.
16) LOCK# is automatically asserted, regardless of the presence or absence of the LOCK prefix.

Appendix A

Programming System Management Mode (SMM)

This appendix provides detailed information on programming the TI486SXL(C) system management mode (SMM). The topics are SMI examples, testing/debugging SMM code, power management features, loading SMM programs, detection of CPU type, presence of SMM-capable devices, creating macros, and altering SMM code limits.

Note:

The final responsibility for verifying designs incorporating TI486SXL(C) microprocessors rests with the customer originating the motherboard design.

Topic **Page**

A.1 SMM Overview

This programmer's guide has been written to aid programmers in the creation of software using the TI486SXL(C) family of microprocessors system management mode (SMM). SMM is currently implemented in all versions of the TI486SXL(C) microprocessors.

A.1.1 Introduction

For an introduction to SMM and additional information, refer to:

- Section A.3, *SMM Features Comparison* (page A-4), which compares the differences between the TI486SXLC and the TI486SXL and other industry offerings that implement SMM
- Subsection A.14.3, *Clearing the VM Flag Bit* (page A-42), which contains important information concerning SMM programming.

A.1.2 SMM Implementation

SMM operation in the TI486SXL(C) microprocessors is similar to related operations in comparable Advanced Micro Devices and Intel Corporation microprocessors. Each implementation:

- Switches into real mode upon entry into the SMM interrupt handler
- Has unique SMM code locations
- Saves the programmer-visible register contents upon entry
- Saves the nonprogrammer-visible register contents

The TI microprocessors have a programmable location and size for the SMM memory region. The TI SMM implementation also provides unique instructions that save additional Segment registers as required by the programmer, in addition to the x86 MOV instruction that saves the General-Purpose registers.

The TI microprocessor automatically saves the minimal register information, reducing the entry and exit clock count to 140. This compares with Intel's clock overhead for entry and exit of 804 clocks and AMD's minimum of 694 clocks. (See Section A.3, *SMM Feature Comparison* (page A-4), for a comparison of SMM overhead.)

Although all three manufacturers' microprocessors provide I/O trapping, the TI486SXL(C) microprocessors SMM simplifies identification of I/O type and instruction restarting. The TI CPU SMM process is unique in its ability to permit software relocation and sizing of the SMM address region. This flexibility facilitates run-time changes to SMM support. This software flexibility lets an operating system or debugger change, modify, or disable the SMM code.

A.2 TI486SXL(C) Microprocessor Power Management Features

The TI486SXL(C) microprocessor family provides several methods and levels of power management. The fully static design, suspend mode, system management mode (SMM), and 3.3-V operation can achieve optimum CPU and system power management. Table A–1 summarizes the various power management options:

Table A–1. Power Management Options

Option	Power Savings
Reduced Clock Frequency	$I_{CC} = (12 \times f_{CLK2\ (MHz)}) + 150$ mA @ 5 V
Lower Supply Voltage (V_{CC})	$I_{CC} = (130 \times V_{CC}) - 256$ mA @ 25 MHz
Suspend Mode	2% of typical I_{CC}
Remove Clock	25% of typical I_{CC}
Suspend Mode and Remove Clock	400 μA
Remove Power	0 μA

A.2.1 Reducing the Clock Frequency

The TI486SXL(C) microprocessor family is a fully static design; the input clock frequency can be reduced or stopped without a loss of internal CPU data or state. The system designer can decide to reduce the clock using SMM capabilities to support advanced power management (APM) software in concert with chipset capabilities. When the clock is removed, then restarted, CPU execution begins with the instruction where the clock was removed. Note that the clock-doubled versions of TI486SXL(C) family must be brought into the non-clock-doubled mode before scaling or stopping the input CLK2.

A.2.2 Suspend Mode

The TI486SXL(C) microprocessor family supports suspend mode operation that can be initiated through software or hardware.

Software initiates suspend mode through execution of a halt (HLT) instruction. After HLT is executed, the CPU enters suspend mode and asserts suspend acknowledge (SUSPA#), if enabled.

Hardware initiates suspend mode by using the SUSP# and SUSPA# pins of the microprocessor. When SUSP# is asserted, the CPU completes any pending instructions and bus cycles and then enters suspend mode. Once in suspend mode, the SUSPA# pin is asserted by the CPU.

A.3 SMM Feature Comparison

The SMM features of the TI486SXLC and TI486SXL microprocessors are compared with other versions of microprocessors in Table A–2.

Table A–2. SMM Features

Feature	TI486SXLC	TI486SXL	386SL	AMD
SMM Entry Point	Base of SMM space (0 to 32M bytes less 4K bytes)	Base of SMM space (0 to 4G bytes less 4K bytes)†	38000h	Reset vector
CPU State Save Area	Top of SMM space	Top of SMM space	3FFA8h–3FFFFh	60000h–600CAh and 60100h–60126h
SMM Space	Programmable (4K to 16M)	Programmable (4K to 4G)	38000/30000h (32K/64K)	Entire address space
Data Auto-Saved	8 32-bit registers 1 16-bit register 1 4-bit register	8 32-bit registers 1 16-bit register 1 4-bit register	44 32-bit registers 9 16-bit registers	53 32-bit registers 8 16-bit registers
SMM Memory Restrictions	None	None	8-bit on 8-MHz XD Bus	Nonpipelined No dynamic bus sizing
Normal Mode SMM Memory Access	Yes	Yes	Yes	No
Hardware Pins	2	2	NA – Must use 82360	4
Incremental CPU State Save Instructions	Yes	Yes	No	No
I/O Trapping	Yes	Yes	Yes	Yes
SMI# Input Masking	Yes	Yes	Yes	No

† Address Region 4 register is 32 bits wide to support 4G-byte physical address space.

A.4 SMM Hardware Considerations

The following sections provide an overview of TI486SXL(C) SMM coding and information helpful in developing SMM code.

A.4.1 SMM Pins

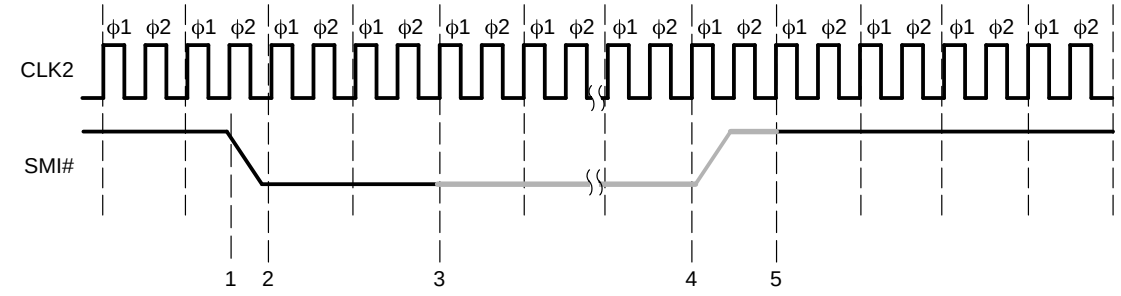
The SMI# and SMADS# pins implement SMM. The bidirectional SMI# pin is used by the chipset to signal the CPU that an SMI has occurred. While the CPU is in the process of servicing an SMM interrupt, the same pin sends a signal to the chipset to indicate that the SMM processing is occurring. The SMADS# address strobe is generated instead of the ADS# address strobe while executing or accessing data in SMM address space.

A.4.2 SMI# Pin Timing

In order to enter the system management mode, the SMI# pin must be asserted for at least four CLK2 periods. See Figure A–1. Once the CPU recognizes the active SMI input, the CPU drives the SMI input low for the duration of the SMI routine. The SMI routine is terminated with an SMI-specific resume (RSM) instruction. When the RSM instruction is executed, the CPU drives the SMI# pin high for two CLK2 periods. The SMI# pin bidirectional design does the following:

- ❑ Prohibits more than one SMI interrupt from becoming active
- ❑ Provides feedback to the chip-set/core logic that an SMI is in process
- ❑ Provides compatibility with other SMM hardware interfaces

Figure A–1. SMI# Pin Timing



A.4.3 Address Strobes

The TI486SXL(C) microprocessor has two address strobes, ADS# and SMADS#. ADS# is the address strobe used during normal operations. The SMADS# address strobe replaces ADS# during SMM operations when data is written, read, or fetched in the SMM defined region. Using a separate address strobe increases chipset compatibility and control.

During an SMM interrupt routine, control can be transferred to main memory via a JMP, CALL, Jcc (conditional jump, cc = condition code) instruction or

execution of a software interrupt (INT). Execution in main memory causes ADS# to be generated for code and data outside of the defined SMM address region. (It is assumed, but not required, that the chipset ultimately translates SMADS# and a particular address to some other address.) To access code in main memory that overlaps the SMM address space, the MMAC bit (CCR1, bit 3) must be set. This allows ADS# strobes to be generated for MOV instructions that overlap main memory while in SMM mode. It is not possible to execute code in main memory that overlaps SMM space while in the SMM mode.

SMADS# can also be generated for memory reads, writes, and code fetches within the defined SMM region when the SMAC bit, Configuration Control 1 register (CCR1) bit 2, is set while in normal mode. (See subsection 2.5.4, *Configuration Registers* on page 2-26, for further information on CCR1). The generation of SMADS# permits a program in normal space to jump into SMM code space. The microprocessor must be in real mode before the jump occurs into SMM space. A routine should be followed to initialize used registers to their real-mode state. The RSM instruction should not be used after jumping into SMM space unless return information is written into the SMM context area before the RSM instruction is executed.

A.4.4 Chipset READY#

The TI486SXL(C) microprocessors have one READY# input. Chipsets that implement the dual READY lines can OR the two ready lines together for the single READY#. The AMD implementation of SMM provides for two READY lines from the chipset, one for SMM space (SREADY#) and one for the normal READY#.

A.5 SMM Software Considerations

At the start of the SMM routine, before control is transferred to code executing at SMM base, some of the CPU state is saved at the end of SMM memory. This is one area where the CPU SMM state is unique. The CPU saves the minimum CPU state information necessary for an interrupt handler to execute and return to the interrupted context. The information is saved at the top of the defined SMM region (starting at SMM base + size – 30h). Of the typically used program registers, only the CS, EFLAGS, CR0, and DR7 are saved upon entry. This requires that data accesses use a CS segment override to save other registers and access data. To use any other register, the SMM programmer must first save the contents using the SVDC instruction for Segment registers or MOV operations for General-Purpose registers (See Section A.7, *SMM Instruction Summary and Macros*, page A-12). It is possible to save all the CPU registers as needed.

The TI486SXL(C) microprocessors are unique in saving the previous IP before the SMI and the next IP to be executed after exiting the SMI handler. Upon execution of an RSM instruction, control is returned to the NEXT IP. The value of the NEXT IP may need to be modified for restarting OUTSx/INSx instructions; this modification is a simple move (MOV) of the PREVIOUS IP value to the NEXT IP location. Execution is then returned to the I/O instruction, rather than the instruction after the next I/O instruction. (The restarting of I/O instructions may also require modifications to the ESI, ECX, and EDI depending on the instruction. See Section A.8, *SMI Handler Example* (page A-17), for typical code used.)

Figure A–2 and Table A–3 describe the SMM header. The P and I bits indicate whether a INSx/OUTSx and a REP prefix were being executed. IN/OUT instructions are restarted by changing NEXT IP and leaving the SMI handler.

Note:

The only area in the SMM header that the programmer should consider altering is the NEXT IP. Altering any other header values can have unpredictable results.

The EFLAGS, CR0, and DR7 registers are set to the reset values upon entry to the SMI handler. This has implications for setting break points using the Debug registers. Break points cannot be set prior to the SMI using Debug registers. The INT 3 debug code trap technique can be used; however, it must be used prior to the occurrence of the SMI in SMM space. Once the SMI has occurred and the debugger has control in SMM space, the Debug registers can be used for the remaining SMI execution.

Figure A–2. SMM Header

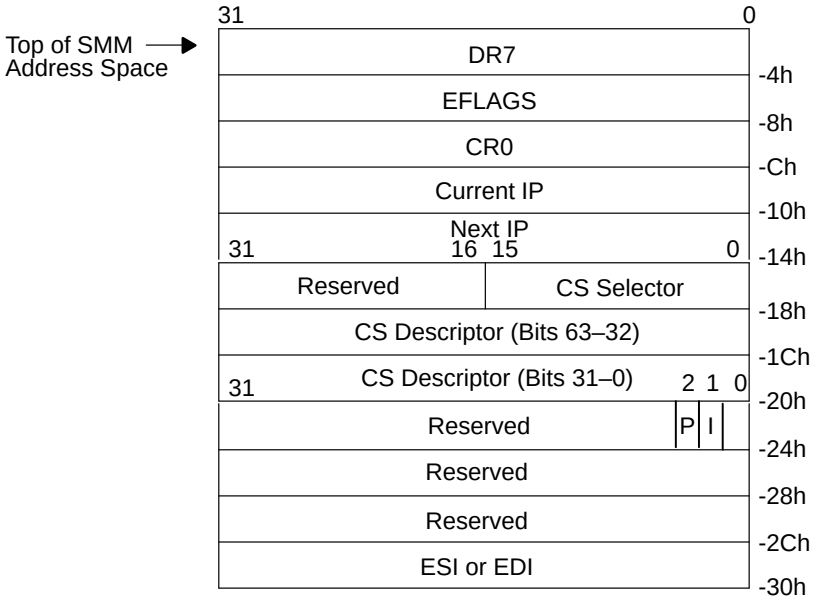


Table A–3. SMM Header

Name	Description	Size
DR7	The contents of the Debug register 7	4 Bytes
EFLAGS	The contents of the extended Flag-Word register	4 Bytes
CR0	The contents of the Control register 0	4 Bytes
Current IP	The address of the instruction executed before servicing the SMI interrupt	4 Bytes
Next IP	The address of the next instruction that will be executed after exiting the SMM mode	4 Bytes
CS Selector	Code Segment register selector for the current code segment	2 Bytes
CS Descriptor	Code register descriptor for the current code segment	8 Bytes
P	REP INSx/OUTSx Indicator: P = 1 if current instruction has a REP prefix P = 0 if current instruction does not have 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
ESI or EDI	Restored ESI or EDI value. Used when it is necessary to repeat an 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, or OUTSD instruction.

A.5.1 Exiting the SMI Handler

When the RSM instruction is executed at the end of the SMI handler, the IP is loaded from the top of the SMM at the address (SMMbase +SMMsize – 14h) called SMI_NEXTIP. This permits the instruction to be restarted. The values of ECX, ESI, and EDI, before execution of the instruction that was interrupted by SMI can be restored from information in the header that pertains to the INx and OUTx instructions. The only registers that are restored from the SMM header are CS, NEXT_IP, EFLAGS, CR0, and DR7.

A.5.2 Accessing Main Memory At the Same Address as SMM Code

To access main memory overlapping the SMM space (i.e., generate ADS# from memory MOV instructions rather than SMADS#), set the MMAC (main memory access) bit in CCR1. The code in Example A–1 enables MMAC:

Example A–1. Accessing Main Memory Overlapping SMM Space

```
       mov    al, 0c1h                                    ;select CCR1
       out    22h, al
       in     al, 23h                                     ;get CCR1 current value
       mov    ah, al                                      ;save it
       mov    al, 0c1h
       out    22h, al
       mov    al, ah
       or     al, 08h                                     ;set MMAC
       out    23h, al
;Now all non–cs–prefixed data memory access will use ADS#;
;Code fetches will continue from SMM memory using SMADS#
;
;Disable MMAC
       mov    al, 0c1h                                    ;select CCR1
       out    22h, al
       mov    al, ah                                      ;get old value of CCR1
       out    23h, al                                     ;and restore it
```

A.5.3 Miscellaneous Execution Details

The following list provides additional details pertaining to the execution of instructions associated with SMM/SMI functions.

- Execution of SMM code begins at the start of SMM space. This is the value entered onto the base portion of AAR4. The CS base is set to the ARR4 SMM base, and EIP is equal to 0. CS limit is the size of the SMM segment set in ARR4.
- The A20# input to the CPU is ignored for all SMM space accesses. These are all accesses that use SMADS#.
- All SMM instructions can be executed outside the SMM defined space, provided that the SMAC bit is set in CCR1 or execution of an SMI handler is in progress. (An SMI handler is in progress during the time the CPU is driving the SMI pin low.)

- Setting the MMAC bit permits the reading and writing of main memory addresses that overlap SMM memory while an SMI is in progress.
- It is not possible to execute code in main memory that overlaps SMM memory addresses while an SMI is in progress.
- NMI is the only enabled interrupt at the entry to the SMI handler. You should provide latches to disable NMI while the SMI is in progress.
- The SMI handler can execute calls, jumps, and other changes of flow and can generate software interrupts and faults using the current definition of the IDT. (Note that on entry to the SMI handler, the IDT is not set to the reset real-mode value of 0:0.)
- The SMI handler can go from real mode to protected mode and vice-versa. Almost anything that can be done normally can also be done during the SMI service routine.
- SMM memory is not cached.
- If the location of SMM space is beyond 1M byte, the value in CS truncates the segment above 16 bits. This prohibits doing calls or INTs from real mode without restoring the 32-bit features of the 486 because of the incorrect return address on the stack.
- An undefined opcode exception is typically generated when conditions are not correct to permit the execution of SMM instructions.
- To execute outside the SMM region (BIOS, debugger, etc.) the CS limit must be changed after entry to the SMI handler. The limit of the CS Segment register is set to the size of the SMM region in ARR4. This means that EIP cannot become larger than the SMM region size. Since jumps in real mode do not change the CS limit, this has implications for software interrupts and jumps out of SMM space. (See Section A.13, *Altering SMM Code Limits* on page A-34 for details and options.)
- Segment registers other than the CS have the limits set in the nonprogrammer-visible portion that were present before the SMI. To avoid a protection error due to the limit or other violations, the RSDC SMM instruction should be used to change the limit of the register in use. (See Section A.12, *Format of Data Used by SVDC/RSDC Instructions* on page A-32.)

A.6 Enabling SMM

To enable and setup SMM in the CPU, set all four of the SMM registers/bits to the values shown in Table A–4 using the code supplied in Example A–2.

See subsection 2.5.4, *Configuration Registers* (page 2-26), for further information on CCR1 and ARR4.

Table A–4.Setting SMM Register Bits

Register/Bit	Location†	Value	Description
SMI	CCR1 bit 1	1	Enable SMI pin
SM4	CCR1 bit 7	1	Make ARR4 as SMM space
SM_loc	ARR4 bits 12–4	Start SMM region	SMM base address
SM_size	ARR4 bits 3–0	≥ 4KB and ≤ 16MB	SMM size

Example A–2. SMM Setup

```
Setup example

        ;SMM Location = 0C8000H
        ;SMM Size = 8KB

        mov     al, 0c1h              ; index to CCR1
        out     22h, al               ; select CCR1 register
        in      ah, 23h               ; read current CCR1 value
        or      ah, 082h              ; enable SMI and SM4 region
        mov     al, 0c1h              ; index to CCR1
        out     22h, al               ; select CCR1 register
        out     23h, ah               ; write new value to CCR1
        mov     al, 0ceh              ; index ARR4 SMM base address bits <23–16>
        out     22h, al               ; select
        mov     al, 0ch               ; set ARR4 SMM base address upper bits
        out     23h, al               ; write value
        mov     al, 0cfh              ; index ARR4 SMM base address bits <15–12>
        out     22h, al               ; and 4 bits for SMM size
        mov     al, 082h              ; set SMM lower address bits and SMM size
        out     23h, al               ; write value
```

A.7 SMM Instruction Summary and Macros

The TI486SXL(C) microprocessor responds to seven instructions when it is in SM mode. These seven nonstandard instructions include:

- Two that save and restore a Segment register and its descriptor
- Two that save and restore the Task register
- Two that save and restore the LDT register
- One that exits SM mode

The instructions that save and restore registers are needed because the CPU saves a minimum amount of information in the SM header (for speed). If one or more of the Segment registers in the SM interrupt handler needs to be modified, the previous values need to be preserved as they are not automatically saved in the header. The instructions that save and restore Segment registers are provided for this purpose. Similarly, the instructions that save and restore the Task register and LDT register allow creation of an SM interrupt handler that enters protected mode and acts as a task dispatcher.

The seven SM instructions summarized in Table A–5 are valid only when CPL is 0 and either of the following is true:

- The SMAC, SMI, and SM4 bits are set and a valid SMM region is defined (the SMM size defined to be greater than 0).
- The SMI# pin is driven low by the CPU. (The CPU drives SMI# low after it recognizes the SMI interrupt and continues to drive it low until RSM is executed. See Figure A–1 page A-5.)

Table A–5. SMM Instruction Set with Clock Counts

Instruction	Mnemonic	Opcode	Clocks	Description
rsdc	rst_seg	0F 79	14	Restores a Segment register from an 80-bit memory location.†
rsldt	rst_ldt	0F 7B	14	Restores the Local-Descriptor-Table register from an 80-bit memory location.†
rsts	rst_tr	0F 7D	14	Restores the Task register from an 80-bit memory location.†
svdc	sav_seg	0F 78	22	Saves a Segment register at an 80-bit memory location.‡
svldt	sav_ldt	0F 7A	22	Saves the Local-Descriptor-Table register at an 80-bit memory location.‡
svts	sav_tr	0F 7C	22	Saves the Task register at an 80-bit memory location.‡
rsm	exit_sm	0F AA	58	Restores the state of the CPU from the data saved in the header at the top of SM memory (the header is created by the processor when it recognizes an SMI). This instruction takes the processor out of SM mode and returns it to the task that was executing when the SMI occurred.

† The restore includes the descriptor information that is invisible to applications.
‡ The save includes the descriptor information that is invisible to applications.

The values in the second column in Table A–5, titled Mnemonic, are arbitrary since there is no current assembler support for the SM instructions. That means that the code is probably generated manually. In generating the code, other arbitrary names may be preferred. The names shown in the first column of Table A–5 are the instruction names that have been added to the TI486SXL(C) instruction set. The mnemonics are a bit more descriptive and are used in the example macros, Example A–3. These examples for generating SM instruction code have been rewritten from earlier versions.

The third column in Table A–5 provides the basic opcode for the SM instructions. In addition to these basic codes, the first six SM instructions listed can be prefixed with a segment override and/or an address size override, and they require a mod r/m byte and a memory offset.

The include file shown in Example A–3 contains some macros that are useful within an SM interrupt handler. These macros implement versions of the seven special SM instructions shown in Table A–5. These macros can be used as is or modified to suit the particular application.

Example A–3. Macros That Implement the Special SM Instructions

```
COMMENT ^
================================================================================
File: SM.MAC

              Copyright (c) 1994 Texas Instruments, Incorporated

This include file defines a set of macros for generating System Management (SM)
mode instruction opcodes, since no assembler directly supports these SM
instructions.

There are six SM instructions that are used to save and restore registers that
are not automatically saved when SM mode is entered, and one instruction for
exiting from SM mode.  These instructions support many addressing modes, but
the macros in this file only implement one mode––a 16–bit memory reference
(within the code segment as a CS: override is also used).  These macros could
be made much more complex to allow other addressing modes, but the additional
complexity wouldn't provide much useful benefit.

Each of the macros that implements a register save or restore takes as a
parameter an offset in the code segment where the register should be saved to
or restored from.  The two macros that save and restore segment registers also
take the name of a segment register as a parameter.

Here is a small portion of code that shows how the macros in this file are used:

<<<<<<< BEGIN EXAMPLE CODE >>>>>>>

.CODE

smi_entry_point:

        sav_seg old_ds,ds                       ; Save segment registers
        sav_seg old_es,es
        sav_seg old_fs,fs
        sav_seg old_gs,gs
        sav_seg old_ss,ss
        sav_ldt old_ldt                         ; Save LDTR and TR
        sav_tr  old_tr
        mov     dword ptr cs:old_eax,eax        ; Save other registers
        mov     cs:old_ebx,ebx

        ...

        rst_seg ds,old_ds                       ; Restore segment registers
        rst_seg es,old_es
        rst_seg fs,old_fs
        rst_seg gs,old_gs
```

```
        rst_seg ss,old_ss
        rst_ldt old_ldt                         ; Restore LDTR and TR
        rst_tr  old_tr
        mov     eax,dword ptr cs:old_eax        ; Restore other registers
        mov     ebx,dword ptr cs:old_ebx
        exit_sm                                 ; Exit SM interrupt handler


old_ds  dt      ?                               ; 10 bytes in code segment
old_es  dt      ?
old_fs  dt      ?
old_gs  dt      ?
old_ss  dt      ?
old_tr  dt      ?
old_ldt dt      ?
old_eax dd      ?
old_ebx dd      ?


...


<<<<<<< END EXAMPLE CODE >>>>>>>


================================================================================
^
; ------------------------------------------------------------------------------
; NOTE: The location at addr must be 10 bytes in size and it must reside
;       within the code segment.  It should be defined as:
;
;addr   dt      ?
; ------------------------------------------------------------------------------


sav_seg MACRO   addr, reg                       ; Save one of the segment registers
        SMMac   sav_seg, addr, reg, 78h
        ENDM


rst_seg MACRO   reg, addr                       ; Restore one of the segment registers
        SMMac   rst_seg, addr, reg, 79h
        ENDM


sav_ldt MACRO   addr                            ; Save the LDT register
        SMMac   sav_ldt, addr, ldt, 7Ah
        ENDM

rst_ldt MACRO   addr                            ; Restore the LDT register
        SMMac   rst_ldt, addr, ldt, 7Bh
        ENDM

sav_ts  MACRO   addr                            ; Save the Task register
        SMMac   sav_ts, addr, ts, 7Ch
        ENDM

rst_ts  MACRO   addr                            ; Restore the Task register
        SMMac   rst_ts, addr, ts, 7Dh
```

```
        ENDM

exit_sm MACRO                                   ; Exit from SM mode
        DB      00Fh, 0AAh


ENDM


SMMac   MACRO   mname, addr, reg, op


        ; CS: override and SM instruction opcode
        db      2Eh
        db      0Fh, op


        ; mod r/m byte
        ifidni          <reg>, <cs>
                db      00Eh
        elseifidni      <reg>, <ds>
                db      01Eh
        elseifidni      <reg>, <fs>
                db      026h
        elseifidni      <reg>, <gs>
                db      02Eh
        elseifidni      <reg>, <ss>
                db      016h
        elseifidni      <reg>, <es>
                db      006h
        elseifidni      <reg>, <ts>
                db      006h
        elseifidni      <reg>, <ldt>
                db      006h
        else
                ECHO ERROR in macro <mname>:
                ECHO Register parameter unknown: <reg>
                ECHO Register parameter must be either CS, DS, ES, FS, GS, SS, TS,
                ECHO or LDT
                .ERR
        endif


        ; 16-bit displacement
        dw      offset addr


ENDM
```

A.8 SMI Handler Example

This section contains fragments of typical coding found in SMI handlers.

Example A–4. Typical Coding Found In SMI Handlers

```
        SMBASE= 0C8000H                         ; base address of SMM space
        SMSIZE= 2                               ; SMM space size is 8k bytes
        SMEND = SMSIZE SHL (SMSIZE–1)           ;works for most cases

        INCLUDE SM.MAC                          ;see Section Example A–3, page A-14
        .MODEL SMALL
        .386P
        .CODE

COMMENT ^
Execution begins here in real mode, with CS defined at the SMBASE and EIP=0
^
        public smi_start
        smi_start:
                jmp     $skipdata               ;skip data area, makes it easy for
                                                ;assembler
        EAXsave         dd      ?
        DSsave          dt      ?
        DStemp          db      0ffh, 0ffh, 0,0,0,92h,8fh,0,0,0  ;4gig present segment
        $skipdata:
                mov             dword ptr cs:[EAXsave],eax; save EAX
                sav_seg         [DSsave], ds         ; save DS
                rst_seg         ds,[DStemp]          ; setDS

COMMENT ^
We need to extend the limits of DS so that we don't get a fault when we use it to ac-
cess low memory. It may be not present with a limit of 0, and these values won't be
changed when we set it using a real mode load.
^


;Determine Why Are We In The SMI Handler

COMMENT ^
Chipset/Core logic unique instructions will follow. The chipset will be used to deter-
mine what caused the SMM interrupt to occur. The BIOS could also "jump" to this point
in the SMM region.


Decision Tree:


        a)      If timer, go to timer_expired

        b)      If port i/o occurred to a trapped location, go to port_io_caused

        c)      If the cpu was turned off, go to cpu_turned_off
^
;timer_expired;
```

```
COMMENT ^
A chipset timer has expired. Unique code would appear to determine which timer. De-
pending on the purpose of the timer, the handler could:

        1)      Reduce the clock frequency
        2)      Execute a halt instruction and enter suspend mode
        3)      Turn current off to the CPU
        4)      Turn off a peripheral device
        5)      Reset the timer and increment a counter

^
reduce_clock:

COMMENT ^
To go to a lower CPU current requirement, the CPU clock can be reduced. The CPU clock
can be reduced from its current setting to a lower value. That value could be zero.
Since the CPU is a static device and will maintain the state of all its registers in
the absence of a clock input there is no state saving requirement. It is assumed that
by writing to the chipset it will reduce or zero the clock. If the clock is stopped,
the next instruction to be executed will be one in this SMI handler immediately fol-
lowing the point where the chipset turned the clock off.
^

        jmp  end_timer:


execute_halt:

COMMENT ^
To go to a lower CPU current consumption, the SMI handler will now execute a HLT
instruction. The HLT instruction will put the CPU into a low power sleep mode until a
non-SMI interrupt occurs. Interrupt(s) will need to be enabled to permit the interrupt
to wake-up the CPU. A common choice would be the keyboard interrupt. A flag would need
to be set in main memory to indicate that the SMI handler should be jumped into or SMI
created, to permit it to restore the state/context of the CPU, prior to the halt for
servicing the interrupt. The interrupt in low memory must point to the BIOS handler
for the return to be made to the SMI handler. An interrupt handler in SMM space could
also service the interrupt rather than a BIOS routine.
^

        ;[ Alternatively the chipset could pull the SUSP# CPU pin low to enter ]
        ;[ suspend mode. The chipset would have to pull SUSP# high to exit ]
        ;[ suspend mode. ]

        :To be sure that BIOS will get control on intr
        ;check for keyboard interrupt vector pointing to BIOS
        ;if not BIOS, save existing and set to BIOS vector or jump to can_not_halt
        ;Set a flag in main memory indicating SMI HALT executed
        ;If an SMM space interrupt handler is used, then IDTR and/or the vector
        ;would need to be updated to the SMM space routine.
        mov ax, 0       ; point to bottom segment
        mov ds, ax      ; ds segment is now in main memory
        mov [485], 1    ; set BIOS flag in main memory
                        ;<set cpu state for bios int>
        hlt             ; last instruction executed here
        ;<the chipset could remove the clock to go to suspend mode now>
        nop

        can_not_halt:  ;CPU state will not be correct at interrupt

        jmp end_timer
```

```
turn_off_cpu:

        ; set bit in main memory to indicate to the BIOS that SMI handler
        ; turned power off to CPU and CPU state should be restored by
        ; the SMI handler
        ;
                mov ax, 0       ; point to bottom segment
                mov ds, ax      ; ds segment is now in main memory
                mov [485], 1    ; set BIOS flag in memory

        ;       (save entire CPU state. See Restore CPU state label)
        ;       (chipset specific instructions to be executed to remove power to
        ;       cpu)
        ;       jmp end_timer

turn_off_peripheral:

        ; chipset specific instructions to turn off peripheral and enable
        ; chipset I/O trapping of the devices io range or enable timer
        ; to allow polling of peripheral requirements.
        jmp end_timer

reset_timer:

        ; chipset specific instructions to be executed to reset a timer and
        ; possibly increment a counter to maintain number to time out
        ; for a particular device.
        jmp end_timer

end_timer:

        jmp done


port_io_caused:

COMMENT ^
The SMM support for I/O being interrupted provides information that permits the re-
starting of the I/O instruction without investigating the actual code where the
instruction is located.

Many things can be done at this point beyond turning on a powered down peripheral. The
CPU clock could now be speeded up in anticipation of heavy CPU processing require-
ments, timers could be reset, etc.
^
        ;** Restart the interrupted instruction

                mov             eax,dword ptr [SMEND+SMI_PREVIOUSIP]
                mov             dword ptr [SMEND+SMI_NEXTIP],eax
                mov             al,byte ptr cs:[SMEND+SMI_BITS]
        ;test for REP instruction
                bt              al,2            ;rep instruction?
                                                ;(result to Carry)
                adc             ecx,0           ;if so, increment ecx
                test            al,1 shl 1      ;test bit 1 to see
                                                ;if an OUTS or INS
                jnz             out_instr
```

```
        COMMENT ^
        ** A port read (INx) instruction caused the chipset to generate an SMI instruc-
tion. Restore EDI saved by SMI microcode.
        ^
                mov             edi, dword ptr cs:[SMEND+SMI_EDIESI]
                jmp             common1
        out_instr:

        COMMENT ^
        ** A port write (OUTx) instruction caused the chipset to generate an SMI
instruction. Restore ESI saved by SMI microcode.
        ^
                mov             esi, dword ptr cs:[SMEND+SMI_EDIESI]
        common1:
                jmp     done


cpu_turned_off:

COMMENT ^
This handler turned off the current to the CPU. Before it did, the handler set a bit
in main memory or battery-backed-up CMOS indicating that this event happened. At re-
set, BIOS will determine that this was the case and ”jump” into the SMI handler. SMI
handler will then restore the entire state/context of the CPU prior to current being
removed. The bit in main memory would also be cleared indicating that the SMI handler
had removed current.
^
        mov ax, 0                       ; point to bottom segment
        mov ds, ax                      ; ds segment is now in main memory
        mov [485], 0                    ; clear BIOS flag in main memory
        mov ax, cs                      ; restore ds to SMM area
        mov ds, ax
```

```
{Restore Complete CPU State}

        ;       eax
        ;       ebx
        ;       ecx
        ;       edx
        ;       edi
        ;       esi
        ;       ebp
        ;       esp
        ;       cs      ;use rst_seg
        ;       ds      ;use rst_seg
        ;       ss      ;use rst_seg
        ;       es      ;use rst_seg
        ;       fs      ;use rst_seg
        ;       gs      ;use rst_seg
        ;       ldtr
        ;       gdtr
        ;       idtr
        ;       tr
        ;       eflags
        ;       cr0
        ;       cr2
        ;       cr3
        ;       dr0
        ;       dr1
        ;       dr2
        ;       dr3
        ;       dr6
        ;       dr7
        ;       ccr0
        ;       ccr1
        ;       ccr2
        ;       Save the configuration registers with index C3h through FFh
        ;       for future product compatibility



       ;      arr1
       ;      arr2
       ;      arr3
       ;      arr4
       jmp done

       done:
              mov      eax,cs:[EAXsave]
              rst_seg  ds,[DSsave]
              exit_sm                   ; return
```

A.9 Loading SMM Memory With an SMM Program From Main Memory

To load SMM memory with an SMI interrupt handler, it is important that the SMI interrupt does not occur before the handler is ready to accept it. This can be done by not having SMAC = 0 and SMI = 1 (in the CCR1 register) before the SMI handler is installed. It is necessary to set SM4 = 1 (in the CCR1 register) and ARR4 with appropriate values before using the SMM memory. ARR4 defines an SMM memory space as a noncacheable memory region when bit SM4 of CCR1 is set to 1.

To load SMM memory with a program, it is first necessary to enable SMM with the exception of the SMI# pin by setting SMAC. (See Section A.6, *Enabling SMM*, page A-11.) The SMM region is then mapped over main memory at the same location. This is done by the generation of SMADS# for memory access for the SMI. A REP MOV instruction can then be used to transfer the program to the location. Then, turn off SMAC to activate potential SMIs.

See Example A–5 for sample code that performs these operations.

Example A–5. SMI Handler Routine

```
.MODEL  MEDIUM

.STACK

; ============================================================================
;                               M A C R O S
; ============================================================================

iodlay_ macro                   ; Short delay for I/O operations
        jcxz    $+2
        jcxz    $+2
endm

segcs_  macro                   ; CS: override prefix
        db      02Eh
endm

include SM.MAC                  ; See Example A–3 page A-14

.CODE


; ============================================================================
;                   S M I   H A N D L E R   R O U T I N E
; ============================================================================

; ----------------------------------------------------------------------------
; When an SM interrupt occurs, the code segment base is set to the SM area
; start as defined in ARR4, and the IP is set to 0.  This means the first SM
; handler instruction must be at offset 0--that is why this loader program
; begins with the SM handler code.  The offsets referenced in the SMI portion
; of this program will be correct in SM mode as well.
; ----------------------------------------------------------------------------
```

```
smi_code_start:


; ------------------------------------------------------------------------------
; Save DS, ES, TS, LDT, AX, and CX (only AX and CX are used by the handler--the
; other registers are only saved to show how the macros are used).
; ------------------------------------------------------------------------------


        sav_seg old_ds, ds
        sav_seg old_es, es
        sav_tr  old_tr
        sav_ldt old_ldt
        mov     dword ptr cs:old_eax, eax
        mov     dword ptr cs:old_ecx, ecx


; ------------------------------------------------------------------------------
; The main handler code goes here ...  The code below simply writes a down
; count to port 80--your code will be much more complex and useful.
; ------------------------------------------------------------------------------


; Write port 80 values
        mov     al, 0FFh
decloop:
        out     80h, al
        mov     cx, 8FFFh
        loop    $               ; Delay
        dec     al
        jnz     decloop


; ------------------------------------------------------------------------------
; Restore registers saved at start of handler, then exit from SM mode.
; ------------------------------------------------------------------------------


        rst_seg ds, old_ds
        rst_seg es, old_es
        rst_tr  old_tr
        rst_ldt old_ldt
        mov     eax, dword ptr cs:old_eax
        mov     ecx, dword ptr cs:old_ecx


        exit_sm                 ; Exit SM mode--resume the interrupted
                                ; program


smi_code_end:


; ------------------------------------------------------------------------------
; The locations below are for saving registers that are used in the SMI routine
; but are not automatically saved when an SM interrupt occurs.  Some of the
; registers saved below are not actually used by the code in this example, but
; they are saved/restored just to demonstrate how the SM macros shown earlier
; are used.
; ------------------------------------------------------------------------------
```

```
old_ds  dt      ?
old_es  dt      ?
old_tr  dt      ?
old_ldt dt      ?
old_eax dd      ?
old_ecx dd      ?


; ===========================================================================
;         P R O C E D U R E S   U S E D   B Y   T H E   L O A D E R
; ===========================================================================


; ===========================================================================
; Read a value from a register in AL via I/O ports 22 and 23.  Return the value
; in AL.
; ===========================================================================
r22_23  proc    near


        out     22h, al
        iodlay_
        in      al, 23h
        ret


r22_23  endp


; ===========================================================================
; Write the value in AH to a register in AL via I/O ports 22 and 23.
; ===========================================================================
w22_23  proc    near


        out     22h, al
        iodlay_
        mov     al, ah
        out     23h, al
        ret


w22_23  endp


; ===========================================================================
;                       L O A D E R   E N T R Y   P O I N T
; ===========================================================================


entry_point:


; ---------------------------------------------------------------------------
; Set ARR4 registers for 64K SMM area at 000A0000:  ARR4 = 000A05
; ---------------------------------------------------------------------------
        mov     ax, 00CDh
        call    w22_23
        mov     ax, 0ACEh
        call    w22_23
```

```
        mov     ax, 05CFh
        call    w22_23


; ---------------------------------------------------------------------------
; Set ARR4 control bit in CCR1 to make ARR4 == SMM memory. Set SMI enable bit
; and SMAC bit to allow non-CS-based data writes to go to the SM area.
; ---------------------------------------------------------------------------
        mov     al, 0C1h
        call    r22_23
        or      al, 86h         ; SM4=1; SMAC = 1; SMI = 1
        mov     ah, al
        mov     al, 0C1h
        call    w22_23


; ---------------------------------------------------------------------------
; Copy SMI code to A000:0000
; ---------------------------------------------------------------------------
        xor     ax, ax
        mov     si, ax          ; SMI code starts at offset 0 of this CS
        mov     di, ax          ; and offset 0 of SM memory too.
        mov     ax, 0A000h      ; SM memory segment
        mov     es, ax
        mov     cx, offset smi_code_end; Number of bytes of SM handler
                                       ; code
        segcs_
        rep     movsb           ; Copy from EXE memory space to SM mem


; ---------------------------------------------------------------------------
; The SM handler is now in place.   Disable access to SM memory leaving the SMI
; bit set, so that SM interrupts can now occur.
; ---------------------------------------------------------------------------
        mov     al, 0C1h
        call    r22_23
        and     al, 0FBh                ; SMAC = 0
        mov     ah, al
        mov     al, 0C1h
        call    w22_23


; ---------------------------------------------------------------------------
; Exit to DOS
; ---------------------------------------------------------------------------
        mov     ax, 04c00h
        int     21h


END     entry_point
```

A.10 Detection of a TI Microprocessor

It is possible, with a small amount of code, to detect if the CPU is a TI microprocessor and if the CPU is the TI486SXL(C) family or a TI486xLC/E family. The following assembler code accomplishes this task.

Example A–6. Detection of a TI Microprocessor

```
;Purpose:      To detect if the CPU is Texas Instruments microprocessor, and then
;              determine if it is a TI486SXLC Family.
;To detect if Texas Instruments:
;              The undefined flags of the TI microprocessor remain unchanged
;              following a divide. An Intel part will modify some of the
;              undefined flags. Check by saving the flags, do a divide,
;              then compare the new flags with the old flags.
;To detect if TI486SXLC Family:
;              The cache test registers in the TI486SXLC Family differ from the
;              TI486xLCE due to the difference in cache size. Bit 9 in TR4 is
;              used to determine if the processor is of the TI486SXLC Family by
;              seeing if it can be toggled.
;              The code that follows is a procedure that returns the CPU detected
;              in AX.

.MODEL SMALL
.486P

;Values that code will return in AX:
CPU_Not TI     EQU             0
CPU_TI486xLCE  EQU             1
CPU_TI486SXLC  EQU             2

TR5_Write      EQU             1
TR5_Read       EQU             2
CR             EQU             0Ah
LF             EQU             0Dh

.CODE
DetectCPU      PROC
StartDetect:
       ;NOTE:
       ;  This procedure returns a value in AX.
       :  Value in BX is destroyed and not saved.
       ;  Value in top–half of EAX is destroyed.
       CLI

AreWeTI486:
       ;Assume that CPU is at least a 386 CPU.
       MOV     AX,     0                       ;set flags to known value
       CMP     AX,     AX

       PUSHF                                   ;save old flags
       POP     AX
       MOV     flags_before,  AX

       MOV     AX, dividend                    ; setup for DIV instruction
       MOV     DX,     0
       MOV     BX,     divisor
       DIV     BX
```

```
        PUSHF                           ;save new flags
        POP     AX
        MOV     flags_after, AX

        MOV     AX, flags_mask          ;isolate bits we are interested in and compare
        AND     AX, flags_before
        MOV     BX, flags_mask
        AND     BX, flags_after

        CMP     AX, BX                  ;flags same before and after?
        JNZ     NotTI                   ;no – don't have TI CPU

WeAreTI486:
        ;Now check to see if CPU is TI486xLCE or TI486SXLC
        MOV     EAX, 0200h              ;attempt to set bit 9 of TR4
        MOV     TR4, EAX
        MOV     EAX, TR5_Write  ;must do write,
                                        ;then read operation on test registers
        MOV     TR5, EAX
        MOV     EAX. TR5_Read
        MOV     TR5, EAX
        MOV     EAX, TR4                ;read TR4 back
        AND     EAX, 0200h              ;isolate bit 9
        CMP     EAX, 0200h              ;did it stay set?
        JNE     FoundTI486SXLC          ;no – found TI486SXLC

FoundTI486xLCE:
        ;CPU is a TI486xLCE
        MOV     AX, CPU_TI486xLCE
        JMP     Done

FoundTI486SXLC:
        ;CPU is TI486SXLC
        MOV     AX, CPU_TI486SXLC
        JMP     Done

NotTI:
        ;CPU is not a TI486
        MOV     AX, CPU_NotTI
        JMP     Done
Done:
        ;leave return value in AX
        RET

DetectCPU       ENDP

.DATA
flags_before    DW              ?
flags_after     DW              ?
flag_mask       DW              08D5h
dividend        DW              0FFFFh
divisor         DW              4h
result          DW              0

END
```

A.11 Detection of SMM Capable Version

At power-up/reset the EDX register contains device identification and stepping information as shown in Table A–6.

Table A–6.EDX Register Data At Power-Up/Reset

EDX	Stepping	SMM Available
0410h	A	No
0421h	B	Yes

The following technique can be used to identify the stepping of a TI486SXL(C) microprocessor after the reset information in EDX is lost.

The method uses two functions: the mixed C and assembler function isb() and the assembly language illegal opcode interrupt handler ill_op. The function isb() returns a 1 to indicate when a B step part is present; it returns a 0 otherwise.

The function isb() installs an illegal opcode handler, ill_op. Then isb() sets up conditions to execute an SMM segment save instruction, SVDC. If an A step part is present, the illegal opcode handler is invoked. The ill_op process then modifies the return address on the stack to return to the instruction after the SVDC instruction. The storage location used by the SVDC instruction is then checked to see if it changed. If it has changed, the part being tested is a B step part. This detection technique must be run at privilege level 0.

Example A–7. Detection of SMM Capable Version

```
//*************************************************************************
//******************************** isb.c **********************************
//*************************************************************************
#define TRUE 1
#define FALSE 0

int old_off;
int old_seg;
extern ill_op();
//*************************************************************************
//                     Function: isb ()
//                     Returns:1 if TI486SXL(C) B step
//                             0 if TI486SXL(C) A step
//*************************************************************************

isb ()
        {
        int i, b_step;
        char mem[10];

        for (i=0; i<10; mem[i++]=0;

        asm     {

                .386
                extrn _ill_op:near
```

```
;*****************************************
;****** get present illegal opcode handler
;*****************************************
       push    es
       push    bx
       mov     ax, 3506h
       int     21h
       mov     old_seg, es
       mov     old_off, bx
       pop     bx
       pop     es

;*****************************************
;****** install new illegal opcode handler
;*****************************************
       push    dx
       push    bx
       push    ds
       mov     ax, 2506h
       mov     dx, OFFSET _ill_op
       mov     bx, cs
       mov     ds, bx
       int     21h
       pop     ds
       pop     bx
       pop     dx

       char save_ccr1, save_cf, save_ce, save_cd;

;*************************************************************
;****** Set SM4 and SMAC and SMI bit to allow SMM instructions
;*************************************************************
       mov     al, 0c1h
       out     22h, al
       in      al, 23h
       mov     byte ptr [save_ccr1, al
       or      al, 86h
       mov     ah, al
       mov     al, 0c1h
       out     22h, al
       mov     al, ah
       out     23h, al

;*****************************************
;****** Setup nonzero SMM region
;*****************************************
       mov     al, 0cfh
       out     22h, al
       in      al, 23h
       mov     byte ptr [save_cf], al
       mov     al, 0cfh
       out     22h, al
       mov     al, 1
       out     23h, al
```

```
;*******************************************
;****** Set SMM region to the top of memory to
;****** avoid overlapping with this program
;*******************************************
        mov     al, 0cdh
        out     22h, al
        in      al, 23h
        mov     byte ptr [save_cd], al
        mov     al, 0ceh
        out     22h, al
        in      al, 23h
        mov     byte ptr [save_ce], al
        mov     al, 0cdh
        out     22h, al
        mov     al, 0ffh
        out     23h, al
        mov     al, 0ceh
        out     22h, al
        mov     al, 0h
        out     23h, al
        mov     al, 0cfh
        out     22h, al
        in      al, 23h
        and     al, 0fh
        out     23h, al

;****** flush prefetch after changing configuration
        jmp     $+2

;*******************************************
;****** Execute SMM instruction sav_seg
;*******************************************
        ;sav_seg word ptr mem, ds
                 Word ptr mem == ss:[bx]
        lea      bx, mem
        db 36h 0fh 78h 1fh

;*******************************************
;****** restore configuration registers
;*******************************************
        mov     al, 0cdh
        out     22h, al
        mov     al, byte ptr save_cd
        out     23h, al
        mov     al, 0ceh
        out     22h, al
        mov     al, byte ptr save_ce
        out     23h, al
        mov     al, 0cfh
        out     22h, al
        mov     al byte ptr save_cf
        out     23h, al
        mov     al, 0c1h
        out     22h, al
        mov     al byte ptr save_ccr1
        out     23h, al
```

```
                ;*******************************************
                ;****** restore old illegal opcode handler
                ;*******************************************
                        push    dx
                        push    bx
                        push    ds
                        mov     ax, 2506h
                        mov     dx, OFFSET old_off
                        mov     bx, OFFSET old_seg
                        mov     ds, bx
                        int     21h
                        pop     ds
                        pop     bx
                        pop     dx
                ) // isb asm region

        for (i=0, b_step=FALSE; i<10; ++i)
                if (mem[i] != 0)
                        {
                        b_step = TRUE;
                        break;
                        }

        return (b_step);
                } // isb ()

;********************** bad_op.asm ***********************
public _ill_op

assumecs:_TEXT

_TEXT segment byte public 'CODE'
_ill_op proc near
                pop   ax
                add   ax, 5
                push  ax
                iret
_ill_op endp
_TEXT ends

end
```

A.12 Format of Data Used by SVDC/RSDC Instructions

The SVDC/RSDC instructions change limits and read/write access privilege levels of the application and System Segment Descriptor registers before they are used by SMM code (see Table 2–7, page 2-22). The instructions use a 10 byte area composed of two major portions of the System Address register set (see Figure 2–7 on page 2-17) value/contents, and the invisible internal descriptor format shown in Example A–8. Example A–9 (page A-33) loads a real-mode system segment (SS) descriptor and invisible region values.

System Segment-Descriptor registers are described in Subsection 2.5.2.2, *Descriptors*, page 2-21.

Example A–8. Internal Descriptor Format

```
|Segment Register Descriptor <8 bytes>|Segment Register Selector <2 bytes>|

;1)     Segment Register Selector: This is the segment if the segment register
;was loaded in real mode or the selector if the segment register was
;loaded in protected mode. In real mode, this is also equal to the segment
;base divided by 10h and clipped to 16 bits.

        dw      |Selector or Segment |

;2)     Segment Register Descriptor, which is the actual descriptor if the
;segment was loaded in protected mode, or a pseudo-descriptor if the segment
;register was loaded in real mode.

        dw      | Limit [15:0] |
        dw      | Base [15:0] |
        db      | Base [23:16] |
        db      | P | DPL | 1 | DscTy[2:0] | A | ; DscTy is descriptor ;type (DT)
        db      |G | D | r | AVL | Limit [19:16] |
        db      | Base [31:24] |
```

Example A–9. Load SS Descriptor Values (Real Mode)

```
;Load SS descriptor with values appropriate to
;REAL mode.

INCLUDE SM.MAC                        ; see Example A–3 page A-14

        old_val         dt      ?       ; location to store old ss value
        real_mode:      dw      0ffffh  ; limit
                        dw      0       ; base
                        db      0       ; base
                        db      10010011B       ; 93h, data segment
                        db      0       ; G=0, D=0, upper limit=0
                        db      0       ; high portion of base
                        dw      0       ; selector/segment

        sav_seg         [old_val], ss
        rst_seg         ss,[real_mode]
        mov             ax, cs
        mov             ds, ax
```

A.13 Altering SMM Code Limits

When the CPU acknowledges an external SM interrupt and switches into system management mode, the CPU is put into real mode. Section 2.8.5, *SMI Service Routine Execution* on page 2-54, states that the Code Segment register is loaded with the base and limits defined by the ARR4 register. If the defined SMM address space is a 16K region, the CS segment limit will be 16K. This is a contradiction to the normal segment limit of 64K for real mode.

This does not normally cause the programmer any problems, since the CS register can access any address in the SMM address space. The only time this can become a problem is if the SMM code jumps to code outside the SMM address space. An example of this might be jumping to a BIOS routine to save a block of memory to the disk drive. The BIOS routine might expect the CS code segment limit to be 64K, and might require it to be, depending on the offset of the routine or any routine it calls. The BIOS procedure might be at offset 38416 of the BIOS segment, for example. If, as stated above, our SMM limit is 16K, then the CPU would generate a segment overrun fault when it attempted to jump to offset 38416 of the BIOS segment.

There are several solutions to this problem. One solution is to never execute code outside of the SMM space. Another solution is to have an SMM space of 64K or larger so that the CS code segment limit is 64K or more. The third solution is to change the CS limits while in the SMM code.

When in real mode, the hidden portion of the Segment registers are not accessible to the programmer, unlike in protected mode. With the new SMM instruction RSDC, a complete 80-bit Segment register and descriptor cache entry can be read from memory into a Segment register, thus changing the segment limits and attributes, even when in real mode. This could be done to make the DS segment have a 4G-byte limit, enabling real mode SMM code to access all of memory with a 32-bit offset, without ever leaving real mode. However, the RSDC instruction will not work with the CS register. The only way to change the limits of the CS segment is to switch to protected mode, do a far jump to a segment descriptor that has the desired segment limit and attributes, and switch back to real mode.

To do this, several things must happen:

1) Setup a GDT with at least one valid entry (this entry is a descriptor for the destination code segment for the intersegment jump).

2) Save the old GDTR register contents (using SGDT), and load the register to point to the new table (using LGDT).

3) Save the old CR0 value, and switch into protected mode with paging off.

4) Do an intersegment jump to the code segment in the GDT, thus changing the CS segment limit.

5) Restore the CR0 value, which switches back to real mode.

6) Restore the saved GDTR value.

A.14 Testing/Debugging SMM Code

There are several ways to debug SMM code:

- Emulation Technology TI486SXLC microprocessor pod with an HP™ 16500/550 Logic Analyzer that provide:
  - Support for selective trace capture
  - SMM instruction disassembly
- Periscope (software only) that provides:
  - Full screen debugging
  - TSR
  - Single stepping and break points
- DOS debug (software only) that provides:
  - Single stepping and break points
- Other selected logic analyzers

A.14.1 Software Only Debugging

It is possible to write an SMI handler and debug it as a TSR. Use a debugger that can set break points at any address in memory. Use the following code sequence, shown in Example A–10, as a model of how to build the SMI handler as a TSR. This code sequence also contains a section that loads the CS section invisible to programmers to change the limit. This is required so that a protection error does not occur when code is executed outside the SMM region. It is assumed that ADS# and SMADS# from the CPU are ORed together by the chipset or external logic. Also, the chipset should support programmable SMM locations.

The code sequence marks the SMI handler address in the user interrupt INT 66 location (0:198h). This lets the programmer determine the location of the SMM region and set break points.

The debugger is able to set a code break point outside the SMI handler using INT 3 only. This is because the Debug (control) register DR7 is set to the reset value upon entry to the SMI handler. This causes break conditions in DR0–3 to be disabled. Debug registers can be used if they are set after entering the SMI handler and saving DR0–3.

Using a TSR to debug SMI has some limitations:

- Other code could overwrite the region.
- Jumps or calls must be to known offsets.

A.14.2 Software Debugging Example

Example A–10 can be used for the first step in debugging SMI code.

Example A–10. Debugging SMI Code

```
        .MODEL SMALL
        .STACK
        .386P
        INCLUDE SM.MAC

RD_WR EQU               12h                     ;read/write
EX_RD EQU               1Ah                     ;execute/readable

COMMENT ^
This is an example of SMI code which can exist below the 1 MByte boundary. It must be
before the 1 MByte boundary because it uses the value in the cs register in order to
form fixups based on its location as well as to jump to return to real mode.
^

        .CODE

smi_handler:
        jmp     $over                           ;pass data area for assembler
        db      100 dup (?)
stacksmilabel
;
;our smi handler gdt
;
gdt     dq      0                               ;null

ADDR = 0
LIMT = 100000h
g_big  = $ – gdt
        dw      (LIMT–1 and 0ffffh)
        dw      (ADDR and 0ffffh)
        db      ((ADDR SHR 16) and 0ffh)
        db      RD_WR OR (0 SHL 5) OR (1 SHL 7)
        db      (((LIMT–1) SHR 16) AND 0fh) OR (0 SHL 6) OR (1 SHL 7)
        db      ((ADDR SHR 24) and 0ffh)
g_code  = $–gdt
ADDR = 0
LIMT = 100000h
        dw      (LIMT–1 and 0ffffh)
        dw      (ADDR and 0ffffh)
        db      ((ADDR SHR 16) and 0ffh)
        db      EX_RD OR (0 SHL 5) OR (1 SHL 7)
        db      (((LIMT–1) SHR 16) AND 0fh) OR (0 SHL 6) OR (1 SHL 7)
        db      ((ADDR SHR 24) and 0ffh)
```

```
GDTSIZE = ($–gdt)

csareadb       10 dup (?)
dsareadb       10 dup (?)
ssareadb       10 dup (?)
esareadb       10 dup (?)
fsareadb       10 dup (?)
gsareadb       10 dup (?)
tsareadb       10 dup (?)

gdtsave df?
gdtnewdw       GDTSIZE – 1
               dd  ?                          ;address

eaxsave dd     ?
ebxsave dd     ?
ecxsave dd     ?
edxsave dd     ?
espsave dd     ?

$over:
COMMENT ^
The debugger may want to use ss,ds,es,fs,gs. The limits may be shortened if the pro-
gram had been running in protected mode. We therefore extend the limits of these reg-
isters before we enable the debugger.
^
        sav_seg [ssarea],ss                   ;save the stack pointer
        sav_seg [dsarea],ds
        sav_seg [esarea],es
        sav_seg [fsarea],fs
        sav_seg [gsarea],gs
        mov     cs:[eaxsave],eax
        mov     cs:[ebxsave],ebx
        mov     cs:[espsave],esp

COMMENT ^
Clear VM flag in Eflags (See Section A.14.3).
^

        rst_seg ss,[gdt+g_big]
        mov     esp, offset smistack
        mov     ax, cs
        mov     ss, ax
        mov     eax, 0
        push    eax
        mov     eax, cs
        push    eax, offset @F
        push    eax
        iretd
@@:
        sgdt    fword ptr cs: [gdtsave]
```

```
COMMENT ^
fixup code for smi base
^
;patch gdt
        mov     eax,cs                          ;segment of us here
        shl     eax,4
        mov     ebx,offset gdt                  ;offset to here
        add     ebx,eax
        mov     dword ptr [gdtnew+2],ebx        ;define gdt base
;patch far jump into protected mode
        mov     ebx,offset $next0
        add     ebx,eax
        mov     dword ptr cs:[patch1],ebx
;patch far jump back to real mode
        mov     word ptr cs:[patch2],cs

start here

COMMENT ^
extend the limits for the code segment

^
        db      66h
        lgdt    fword ptr [gdtnew]
        mov     eax,cr0
        or      al,1
        mov     cr0,eax
        db      66h
        db      0eah
patch1  dd      ?
        dw      g_code

$next0: mov     bx,g_big                        ;extend the limits of the data segments
        mov     ss,bx
        mov     ds,bx
        mov     es,bx
        mov     fs,bx
        mov     gs,bx
        xor     al,1
        mov     cr0,eax                         ;back to real mode
        db      0eah
        dw      offset $next1
patch2  dw      ?                               ;far jump to set cs and writable bit
$next1:
```

```
COMMENT ^
define a valid stack
^
        mov     ax,cs
        mov     ss,ax
        mov     esp,offset stacksmi
COMMENT ^
****** Insert user specific smi code here & set breakpoints. ******
^
        db      66h
        lgdt    fword ptr cs:[gdtsave]
        rst_seg ss,[ssarea]
        rst_seg ds,[dsarea]
        rst_seg es,[esarea]
        rst_seg fs,[sarea]
        rst_seg gs,[gsarea]
        mov     eax,dword ptr cs:[eaxsave]
        mov     ebx,dword ptr cs:[ebxsave]
        mov     esp,dword ptr cs:[espsave]
        exit_sm
smi_handlere:
SMI_SIZE = offset smi_handlere – offset smi_handler
Install PROC

;***** Enable SMM Region ******
; Don’t enable SMI yet because we’re not ready for it.
        mov     al, 0c1h                ;select CCR1
        out     22h,al
        in      al, 23h                 ;read CCR1
        or      al, 80h                 ;enable SMADS# and SMM region (not SMI)
        mov     ah, al
        mov     al, 0c1h                ;select CCR1
        out     22h, al
        mov     al, ah
        out     23h, al                 ;write new CCR1 value

        mov     eax,offset endresident
        mov     ebx,cs
        shl     ebx,4
        add     eax,ebx
        add     eax,0fffh
        and     eax,NOT 0fffh           ;eax = start of smi space
        mov     edx,eax
        push    edx
```

```
;*****************************************************************************
; * Load SMI address and size into ARR4
;
;******                      cd         ce            cf
;******                 -----------  -----------  ------------
;****** Config Reg      31–28 27–24, 23–20 19–16, 15–12 <size>
;****** Address         31–28 27–24, 23–20 19–16, 15–12 11–8,  7–4 3–0

        mov     al, 0cdh                ;region 4 1st word
        out     22h, al
        mov     eax, edx                ;get smi handler address
        shr     eax, 24                 ;move address <31–24> to al
        out     23h, al                 ;[7–0]=>smbase[31–24]

        mov     al, 0ceh                ;region 4 2nd word
        out     22h, al
        mov     eax, edx                ;get smi handler address
        shr     eax, 16                 ;move address <23–16> to al
        out     23h, al                 ;[7–0]=>smbase[23–16]

        mov     al, 0cfh                ;region 4 3rd word
        out     22h, al
        mov     eax, edx                ;get smi handler address
        shr     eax, 8                  ;move address <15–12> to al
        and     al, 0f0h                ;clear bottom nibble
        or      al, 1                   ;select 4KB SMI size
        out     23h, al                 ;and [3–0]=>smsize
;*****************************************************************************
        pop     edx                     ;start of smi area
        mov     eax,edx
        add     edx,1000h               ;reserve 4k for smi handler
        mov     ebx,es                  ;current psp
        shl     ebx,4                   ;
        sub     edx,ebx                 ;bytes to reserve
        she     edx,4                   ;paragraphs to reserve in dx
        push    dx
        shr     eax,4                   ;paragraph of smi handler
        mov     es,ax                   ;save for later
        mov     ds,ax
        mov     dx,0                    ;always starts at 0
        mov     ax, 2566h               ;int 66h vector at 0:198h
        int     21h
        pop     dx                      ;tsr address
```

```
;move the code to the smi_area
        mov     al, 0c1h                ;select CCR1
        out     22h, al
        in      al, 23h                 ;read CCR1
        mov     ah, al                  ;save old value
        mov     al, 0c1h                ;select CCR1
        out     22h, al
        mov     al, ah                  ;get old value
        or      al, 04h                 ;enable SMAC
        out     23h,al                  ;be clean on ah for later
RELOCATE = 0
IF RELOCATE
        sub     esi,esi
        sub     edi,edi
        mov     cx,cs
        mov     ds,cx
        mov     ecx, (SMI_SIZE+3)/4
        rep     movs dword ptr es:[edi],dword ptr ds:[esi]
ELSE
;put the far jump at the start of the smi_area to above code
        mov     byte ptr es:[0],0eah
        mov     word ptr ex:[1],offset smi_handler
        mov     word ptr ex:[3],cs
ENDIF
;restore smi state and enable SMI
        mov     al, 0c1h                ;select CCR1
        out     22h, al
        mov     al, ah                  ;get old value
        or      al, 02h                 ;set SMI bit to enable SMI
        out     23h,al                  ;be clean on ah for later
COMMENT ^
SMIs may happen at any time now.
^
;dx = offset in this segment to tsr
        mov     ax, 3100h               ;Request function 31h, error code=0
        int     21h                     ;Terminate–and–Stay–Resident
Install ENDP
;----end of resident code----
endresident     label byte

        db  2000h  dup (?)

        END     Install

;******************************************************************************
```

A.14.3 Clearing the VM Flag Bit

If the CPU is in V86 mode and is interrupted by an SMI, the VM bit in the EFLAGS register is not cleared as it should be during real-mode operation. Not clearing this bit can cause protection errors of valid instructions that are being executed in the SMI handler. This can be resolved by adding the following code after saving all used registers:

```
rst_seg ss, [gdt+g_big]        ; change ss limit to 4 Gbytes
mov     esp, offset smistack           ; create new stack pointer
mov     ax, cs
mov     ss, ax                         ; new stack segment
mov     eax, 0
push    eax                            ; flags after iretd
mov     eax, cs
push    eax                            ; segment after iretd
mov     eax, offset @F
push    eax                            ; offset after iretd
iretd
     @@:
```

See the debugging example in Section A.14, *Testing/Debugging SMM Code,* for usage of this code.

Appendix B

BIOS Modifications Guide

To reap full benefit from the TI486SXL(C) family of microprocessors, the system BIOS should be modified to support the internal registers that control the on-chip cache, clock doubling, and other features. This appendix serves as a guide to some of the changes that need to be considered, and includes sample assembler code for controlling the cache.

The following three topics are discussed regarding the internal cache registers and clock double enable:

- Power-up and hard reset
- Protected-mode to real-mode switching
- Soft reset— CONTROL ALT DELETE

In each case, the state of the CPU cache registers and the clock-double enable bit must be known to determine when and how to change their values.

Note:

The final responsibility for verifying designs incorporating TI486SXL(C) microprocessors rests with the customer originating the motherboard design.

Topic **Page**

B.1 Differences Between the TI486SLC/DLC BIOS and the TI486SXL(C) BIOS

The TI486SLC/DLC BIOS requires some modifications to support the new features of the TI486SXL(C) family of microprocessors.

If the BIOS currently tests the internal cache before enabling it, the test routine requires modification. Due to the larger size of the TI486SXL(C) cache, the cache test registers have changed from those in the TI486SLC/DLC. (See Table 2–17 on page 2-36.) It is unnecessary to test the TI486SXL(C) cache before enabling it during the boot process.

In addition to changing the cache test registers, the cache organization selection bit has been redefined. In the TI486SLC/DLC, configuration control register 0 (CCR0) bit 6 is used to select between a direct-mapped and a two-way, set-associative, internal cache organization. For the TI486SXL(C) family, the cache is always two-way set associative and CCR0 bit 6 is defined to enable clock-doubled mode. BIOS prepared to support the TI486SLC/DLC can allow the user to select the cache organization, but BIOS prepared for the TI486SXL(C) should comprehend that the cache-organization selection is not available.

If the BIOS supports software clock switching, a modification to support clock-doubled feature may be desirable. Switching to high-speed mode should enable bit 6 of CCR0 and thus put the CPU in clock-doubled mode. Switching down the CPU speed should disable bit 6 of CCR0 and put the CPU in nonclock-doubled mode. If the BIOS is APM (advanced power management) compliant, the use of 1x and 2x modes should be implemented as well.

Note:

When the TI486SXL(C) is in clock-doubled mode, the CLK2 input must not be scaled or stopped. First, the processor must be placed in nonclock-doubled mode; then, the CPU clock speed can be changed.

When the TI486SXL(C) family microprocessors are reset, the cache and the clock-doubled features are disabled by default.

B.2 Power-Up and Hard Reset

During power-up and hard reset, the operating system (OS) is booted up. Due to the reset line to the CPU going active, the internal cache and the clock-doubled feature are disabled, making the CPU act similar to a 386. If the cache and the clock-doubled feature are enabled prior to the reset, they must be turned on at some point before the OS is booted. A convenient time may be during final chipset initialization, understanding that the cache should remain off during memory sizing. Many BIOSs provide the user an option to disable the system cache using the setup screen. Because most user cache-control options are stored in nonvolatile RAM, the flag responses and potentially other flags should be checked before turning the cache on.

B.3 Protected-Mode to Real-Mode Switching

Protected-mode to real-mode switching can be implemented to handle cases where the OS has been booted, applications are running, and the CPU needs to be reset from protected to real mode. The object is to switch CPU modes and jump back into the OS or application at some saved return address.

When the CPU is reset, the internal cache and the clock-doubled feature are disabled. Before returning control to the application, the cache and clock doubling should be turned back on, but only if they were enabled before the reset occurred. This is accomplished by checking the cache-enable flag in the nonvolatile RAM to see if the user enabled caching from the setup screen. However, if the BIOS allows the user to turn off the cache by a hot-key combination (perhaps as part of speed switching), other checks may need to be performed to see if the cache should be turned back on.

B.4 Soft Reset

Execute soft reset by pressing CONTROL ALT DELETE. The objective of a soft reset is to reset the system and reboot the OS. The action is similar to power-up and hard reset, but a hard reset of the CPU is not generated. Thus, the CPU's internal cache and clock doubling are not disabled. Since the cache is not disabled, this soft reset can negatively impact memory-sizing code, such as generating memory-size mismatch errors. In this situation, disable the internal cache and enable it prior to booting if it was enabled by the user in setup.

B.5 Turning the Internal Cache On and Off

When the TI486SXL(C) family of microprocessors internal cache is turned on or off, the following guidelines should be observed in the order presented:

1) Turn off interrupts—CLI

2) Turn off cache using Control Register 0 (CR0) bit 30 and flush using WBINVD

3) Manipulate cache registers

4) Turn on cache and flush using WBINVD

5) Turn on interrupts—STI

This sequence ensures that the process is not interrupted until complete and that no cache coherency issues arise when the cache is turned back on. When manipulating the cache registers, set each register explicitly instead of relying on default values.

Example B–1 shows assembler code for turning the cache off.

Example B–1. Turning Internal Cache Off

```
CacheOut      MACRO               index, value
              MOV                 AL, index
              OUT                 22h, AL
              MOV                 AL, value
              OUT                 23h, AL
CacheOut ENDM

CLI
MOV           EAX, CR0
OR            EAX, 40000000h            ; set bit 30, turn off cache
MOV           CR0, EAX
WBINVD                                  ; for external cache coherency

CacheOut      0C0h,       00h
CacheOut      OC1h,       00h

CacheOut      0C4h,       00h
CacheOut      0C5h,       00h
CacheOut      0C6h,       0Fh

CacheOut      0C7h,       00h
CacheOut      0C8h,       00h
CacheOut      0C9h,       00h

CacheOut      0CAh,       00h
CacheOut      0CBh,       00h
CacheOut      0CCh,       00h

CacheOut      0CDh,       00h
CacheOut      0CEh,       00h
CacheOut      0CFh,       00h

WBINVD
STI
MOV           EX,         4C00h
INT                       21h           ; return to DOS
```

Turn on the microprocessor internal cache by modifying some of the register values as shown in Example B–2. The CacheOut macro definition remains the same.

Example B–2. Turning Internal Cache On

```
CLI
MOV          EAX, CR0
OR           EAX, 40000000h             ; set bit 30, turn on cache
MOV          CR0, EAX
WBINVD                                  ; for external cache coherency

CacheOut     0C0h,       23h            ; set bits NC1, NC0, BARB
CacheOut     OC1h,       00h

CacheOut     0C4h,       00h
CacheOut     0C5h,       00h
CacheOut     0C6h,       00h

CacheOut     0C7h,       00h
CacheOut     0C8h,       00h
CacheOut     0C9h,       00h

CacheOut     0CAh,       00h
CacheOut     0CBh,       00h
CacheOut     0CCh,       00h

CacheOut     0CDh,       00h
CacheOut     0CEh,       00h
CacheOut     0CFh,       00h

MOV          EAX, CR0
AND          EAX, NOT 40000000h
MOV          CR0, EAX                   ; clear CD bit
WBINVD
STI
MOV          EX,4C000h
INT          21h                        ; return to DOS
```

Appendix C

Design Considerations and Cache Flush

This appendix discusses design considerations, address bit A20 masking, and general cache invalidation procedures.

Note:

The final responsibility for verifying designs incorporating TI486SXL(C) microprocessors rests with the customer originating the motherboard design.

Topic **Page**

C.1 Design Conventions

Following these conventions to connect the TI486SXL(C) terminals to the PCB:

- ❑ Connect (short) all V_{CC} terminals to the positive supply voltage.
- ❑ Connect (short) all V_{SS} (GND) terminals to the system ground.
- ❑ For the TI486SXL in the 144-pin package connect (short) both W/R# terminals (terminals 36 and 37) together and connect to W/R# signal source.
- ❑ Leave electrically open (unconnected) all NC terminals.

Note:

Connecting or terminating (high or low) any NC terminal(s) can cause unpredictable results or nonperformance of the microprocessor.

C.2 Address Bit A20 Masking

The A20M, address bit 20 mask, is an anomaly in PC designs resulting from the fact that truncated addresses can be generated by an 8086/8088 outside the physical address range of 0h–FFFFFh. For example, an 8086/8088 system that contains FFFFh in a segment register and 0FFFh in an offset register results in an address of 100FFEh that requires 21 bits to address. Since the 8086/8088 has only 20 address bits (A0–A19), the most significant bit of the resultant address would need to appear on an A20 bit if the 8086/8088 had one.

Since the 8086/8088 address bus is not wide enough, only the first 20 bits of the address are seen by the system. Using the address 100FFEh, generated in the previous example, the 8086/8088 system read/write address is performed at location FFEh and not at 100FFEh. The 80286 and later microprocessors implement at least 24 address bits and perform the read/write to address location 100FFEh. Thus, software applications can produce different results when run on an 8086/8088 system versus an 80286 or later microprocessor system.

Systems that use 80286 or later microprocessors compensated for this anomaly by adding circuits to generate an A20 mask (referred to as the A20 mask or the A20 gate, or similar). The A20 mask consists of software-controlled logic that forces a zero on the A20 address line regardless of the actual value of A20. The software-controlled A20 mask can also permit the true value to be passed to the system when required.

Note that the A20 mask logic is external to the processor in both 80286 and 80386 designs. The processor generates the actual address but the system logic can be set to ignore or not ignore the A20 pin. Normally, the A20 pin is ignored when these processors are executing in real mode and emulating an 8086/8088.

This is an important consideration when replacing an 80386SX/DX device with a TI486-type device. The TI486SXL(C) microprocessors implement an internal cache, and, if the system is in a state that ignores the A20 address input, the processor must know so that it can also ignore the A20 address input.

If the A20M bit of configuration control register 0 (CCR0) is set, the TI486SXL(C) microprocessor knows that the A20M input provides the true value required. However, if the TI486SXL(C) is inserted into a socket designed for the 80386SX/DX, the TI486SXL(C) A20M pin is placed at a pin location that is not used by the 80386SX/DX. The system hardware needs to be modified to provide the A20M connection.

The NC0 bit of CCR0 is a software-only solution to the A20 mask function. When set, the TI486SXL(C) microprocessor does not cache the first 64K bytes of memory above each 1M byte boundary. This solution means that, even if the value of the A20 address is not known, the processor does not cache data to the affected addresses.

C.3 General Cache Invalidation

When the FLUSH bit in configuration control register 0 (CCR0) is set, the FLUSH# input, when asserted low, invalidates the contents of the TI486SXL(C) internal cache. This can be used to assure that data stored in the TI486SXL(C) internal cache does not differ from data stored in system memory. Additionally, the cache can be invalidated by execution of the 486-compatible invalidate instructions (INVD,WBINVD) or in response to a hold acknowledge state if the BARB bit in CCR0 is set. The method chosen for invalidating the TI486SXL(C) internal cache can be different, depending on whether the system has a serial secondary cache. Invalidation methods are described for systems with and without a serial secondary cache.

C.3.1 Systems With No Secondary Cache or With a Parallel Secondary Cache

When the only cache memory in the system is the TI486SXL(C) internal cache, or when the secondary cache has a parallel (or look-aside) architecture, there are two general methods of invalidating the cache and maintaining cache coherency.

C.3.1.1 Invalidation Method 1

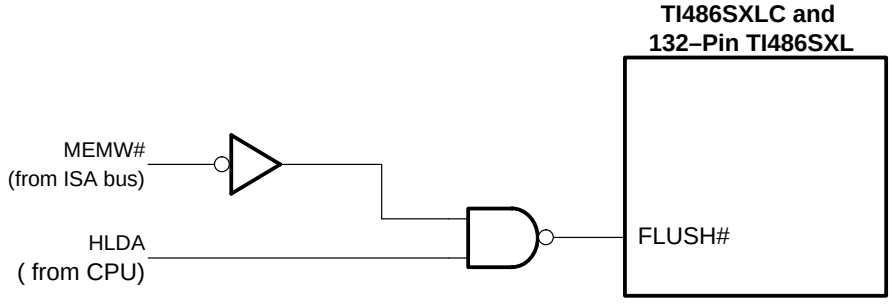
Invalidate the TI486SXL(C) cache every time the CPU enters a hold state. By setting the BARB bit in CCR0, automatic cache flush occurs when the TI486SXL(C) is placed in a hold state. If the chipset does not support hidden refresh, very frequent cache invalidation may occur since the CPU is placed in hold during DRAM refresh cycles that occur approximately every 15 μs. If the chipset supports hidden refresh, this may be an acceptable solution since the cache is only invalidated during DMA or bus master reads from or writes to memory.

C.3.1.2 Invalidation Method 2

Invalidate the TI486SXL(C) internal cache when a DMA or bus master writes to system memory. The external hardware must drive the TI486SXL(C) FLUSH# or MEMW#† input when DMA or bus masters are detected writing to system memory. This can be done using one of the circuits shown in Figure C–1 or Figure C–2.

Figure C–1 shows the circuitry needed to generate an active-low FLUSH# to the CPU each time a hold state is entered (defined by HLDA = 1) and memory write occurs (defined by MEMW# = 0).

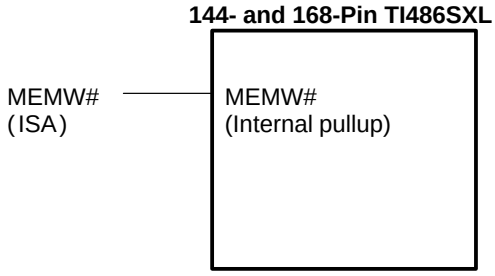
Figure C–1. Cache Invalidation for the TI486SXLC and the 132-pin TI486SXL



† MEMW# input is implemented on the 144-pin and 168-pin TI486SXL only.

The 144-pin QFP and 168-pin PGA versions of the TI486SXL have the external hardware shown in Figure C–1 incorporated on chip. Therefore, to maintain cache coherency in these two devices, connect the MEMW# signal from the ISA bus to the MEMW# input as shown in Figure C–2.

Figure C–2.Cache Invalidation for the 144- and the 168-Pin TI486SXL

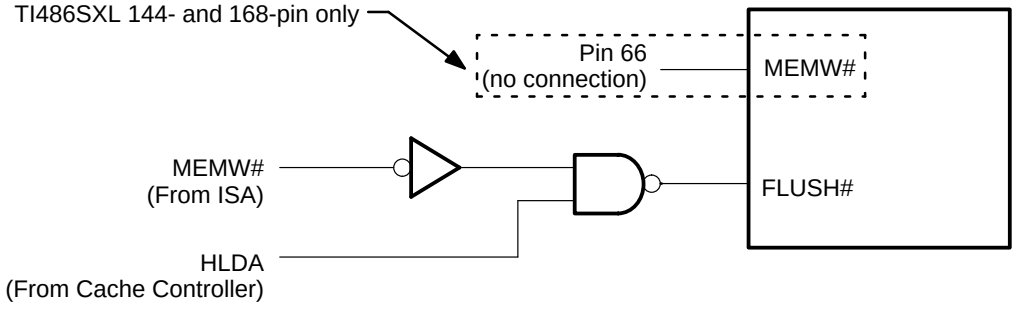


C.3.2 Systems With a Serial Secondary Cache

In a system with a serial (or look-through) secondary cache, flushing the cache cannot be accomplished by setting the BARB bit in CCR0. Bus arbitration occurs between the serial cache controller and the system, allowing the CPU to continue executing out of cache.

The secondary cache controller arbitrates the bus among itself and DMA controllers or bus masters and asserts HLDA to the chipset when the bus has been granted. Each time a DMA or bus master write is detected, the FLUSH# pin on the TI486SXL(C) must be asserted. The circuit shown in Figure C–3 can be used. Note that the HLDA signal is generated by the secondary cache controller rather than the CPU. This is the preferred solution since, in many cases with secondary serial caches, the CPU is not put in hold so it can continue execution from cache while DMA or bus-master activity occurs on the system bus.

Figure C–3.FLUSH# Logic With a Serial Secondary Cache



Appendix D

OEM Modifications for 168-Pin CPGA

This appendix describes the potential modifications an original equipment manufacturer (OEM) needs to implement on an existing 486SX/DX motherboard to take advantage of the TI486SXL 168-pin CPGA. This package offers OEMs added flexibility in implementing solutions that support various 486 CPUs with the same motherboard.

The pinout of the TI486SXL 168-pin CPGA is nearly identical to the Intel™ or AMD™ 486SX CPGA pinout. The NC pins on the TI486SXL package that match signal pins on the 486SX have no internal connection and can be left connected to the 486SX signal pins when the board is configured as a TI486SXL board. This greatly simplifies the interface for the OEM. The classes of board designs covered are listed in the topic index below.

The board design requires the use of system logic that supports the Intel/Advanced Micro Devices 486 interface and the TI486SXL interface. Since board modifications for TI486SXL support are system-logic dependent, the implementation details are left to the board designer.The design examples show both *required* and *optional* jumper connections that *can* be made if the functions associated with them are needed. None of the optional signals require termination if not used.

Subsection D.5, *Power Planes for 3.3-V and 3.3-V/5-V Systems Using TI486SXL or 486DX4* on page D-9, shows 2 system implementations:

- A 3.3-V system that supports a 5-V ISA and a 3.3-V VL bus.
- A mixed 3.3-V/5-V system that supports a 5-V ISA and a 5-V VL bus.

In both implementations the microprocessor runs at 3.3 V.

Note that you have the final responsibility for verifying designs incorporating any version of a TI486SXL microprocessor.

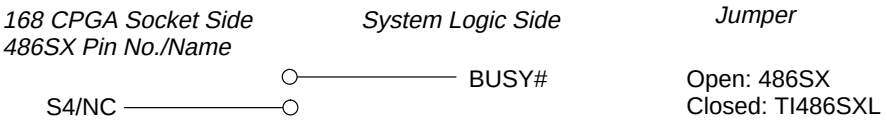
Topic **Page**

D.1 Boards Supporting TI486SXL and Intel

Pin names and assigned locations are provided in Chapter 6, *Mechanical Specifications.*

- Function: Connect BUSY# to S4 (Required)

  BUSY# is required for the coprocessor and for self test. If neither is used, BUSY# can be left open as it has an internal pullup resistor.



- Function: Hardware Cache Flush Support

  - CASE 1: Systems with no level-2 or parallel cache (optional)

    Hardware flush support for the TI486SXL is optional as this function may be implemented in software by setting bit 5 in TI486SXL Configuration Control register 0 (CCR0). However, the software implementation may negatively impact the performance of certain designs. To achieve maximum system performance, a hardware implementation is recommended as illustrated in Figure D–1. Also, see Appendix C, *Design Considerations and Cache Flush*, for more information.

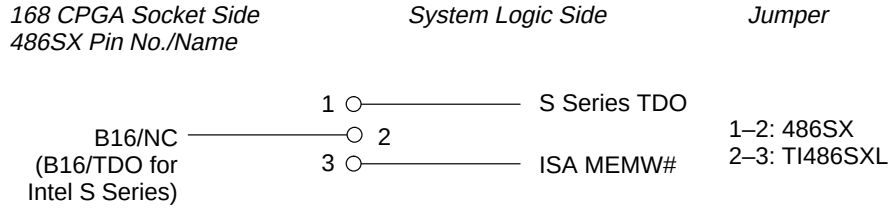
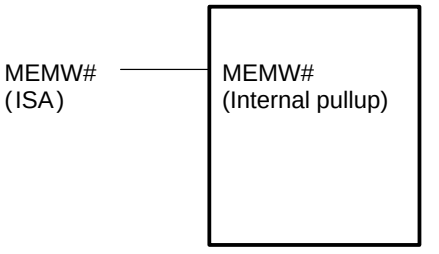


Figure D–1.FLUSH# for 144-Pin and 168-Pin TI486SXL



Note: The external flush logic is incorporated on the 144-pin and 168-pin TI486SXL chip.

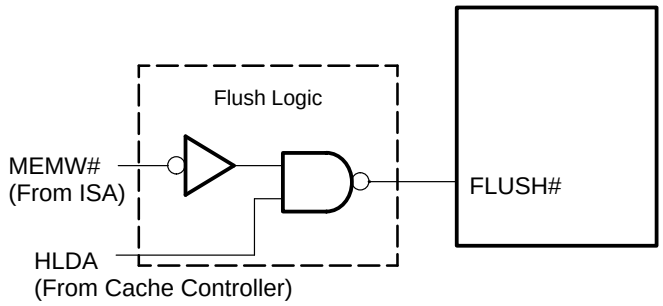
Or

- CASE 2: Systems with a level-2 serial cache that do not hold the CPU during all DMA/Master cycles (required)

168 CPGA Socket Side 486SX Pin No./Name	*System Logic Side*	*Jumper*
C15/FLUSH# — 1	SX FLUSH#	1–2: 486SX
C15/FLUSH# — 2		2–3: TI486SXL
C15/FLUSH# — 3	TI486 FLUSH#	

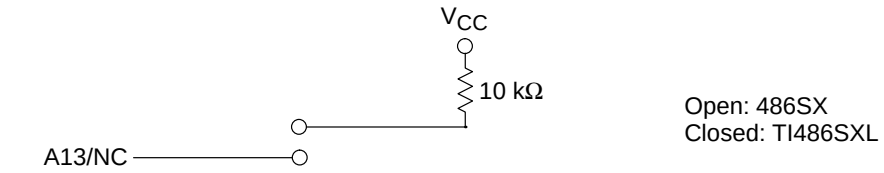
The FLUSH# hardware implementation is shown in Figure D–2.

Figure D–2.FLUSH# Logic With Level-2 Serial Cache

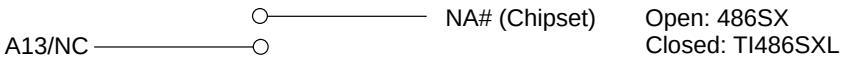


❑ Function: Pipeline Support (Required)

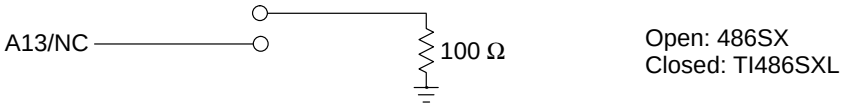
■ CASE 1: Chipset does not support pipelining.



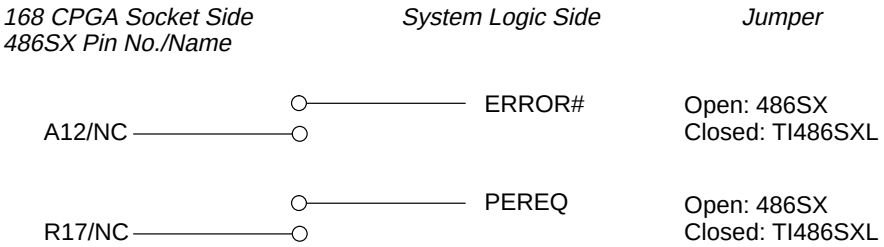
■ CASE 2: Chipset supports pipelining and drives NA#.



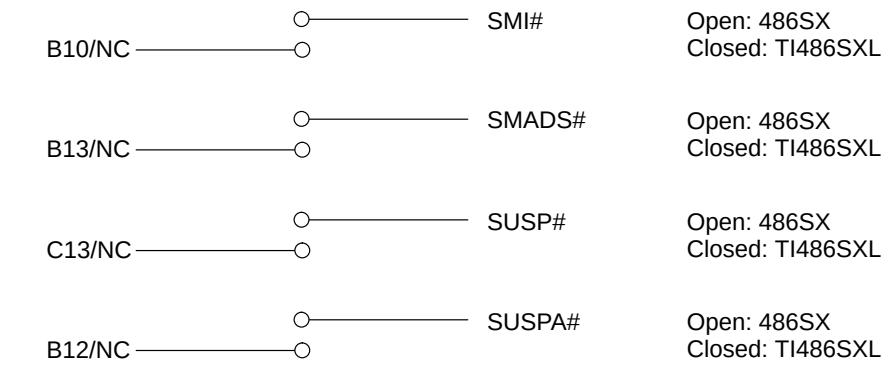
■ CASE 3: Chipset supports pipelining but does not drive NA#.



❑ Function: Floating Point Unit (FPU) Support (Optional)



❑ Function: Power Management Support (Optional)



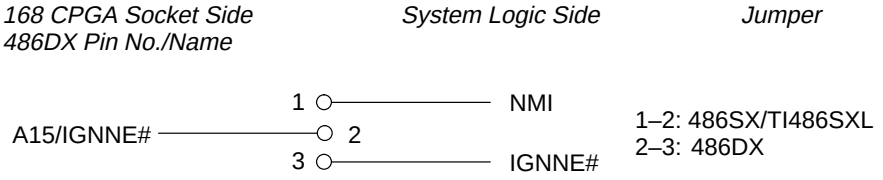
D.2 Boards Supporting TI486SXL and a 486DX

Pin names and assigned locations are provided in Chapter 6, *Mechanical Specifications.*

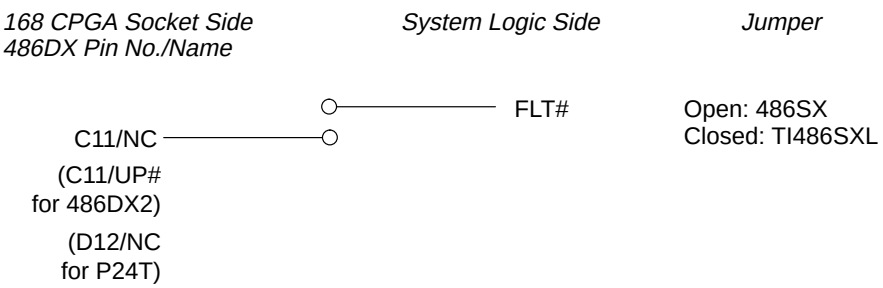
❑ Function: 486DX Support (Required)

Note:

For the 486DX to be supported in the same design, the following jumper is required in addition to those shown in Section D.1, *Boards Supporting TI486SXL, Intel, and AMD 486SX*, and any other differences in Intel-/AMD-supported pinouts.



❑ Function: 486DX2, P24T Upgrade Socket Support (Optional)



D.3 Boards Supporting TI486SXL and a 486DX4

Pin names and assigned locations are provided in Chapter 6, *Mechanical Specifications.*

❑ Function: 486DX4 PEREQ and CLKMUL (Required)

Note:

For the TI486SXL and the 486DX4 to be supported in the same design, the following jumpers are required in addition to any other differences in Intel-/AMD-supported pinouts. See subsections D.4, *Boards Supporting the VL Bus* on page D-7, and D.5, *Power Planes for 3.3-V and 3.3-V/5-V Systems Using TI486SXL or 486DX4* on page D-9.

168 CPGA Socket Side 486DX4 Pin No./Name	*System Logic Side*	*Jumper*
R17/CLKMUL — 1	PEREQ	1–2: TI486SXL 2–3: 486DX4
R17/CLKMUL — 2		
R17/CLKMUL — 3	CLKMUL	

❑ Function: Voltage Detect (Required)

168 CPGA Socket Side 486DX4 Pin No./Name	*System Logic Side*	*Jumper*
S4/VOLDET — 1	BUSY#	1–2: TI486SXL 2–3: 486DX4
S4/VOLDET — 2		
S4/VOLDET — 3	VOLDET	

❑ Function: Burst Mode (Required)

168 CPGA Socket Side 486DX4 Pin No./Name	*System Logic Side*	*Jumper*
10 kΩ resistor — 1	V_{SS}	1–2: TI486SXL 2–3: 486DX4
2	BLAST# (VL slot)	
R16/BLAST# — 3		

D.4 Boards Supporting the VL Bus

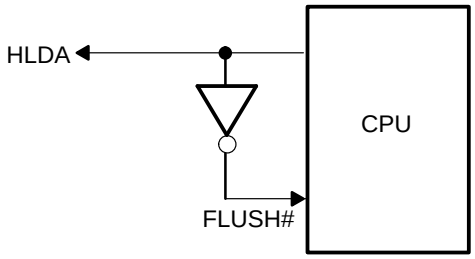
In order to support the VESA VL bus 2.0p proposal, the following design guidelines should be considered.

D.4.1 Cache Snooping

In a VL-bus design, the local bus controller resolves arbitration between the CPU and the VL-bus master. For this architecture, the CPU can be forced to relenquish the host bus by asserting HOLD. There are two options for maintaining cache coherence:

- Use the BARB bit in Configuration Control register 0 (CCR0) to flush the internal cache.
- Use the inverted HLDA output of the CPU to perform a hardware FLUSH# to the CPU. See Figure D–3. The FLUSH# pin must be enabled by using bit 4 of CCR0.

Figure D–3.Hardware Flush



Note: Pin names and assigned locations are provided in Chapter 6, *Mechanical Specifications.*

These methods can be used only if the system logic supports the CPU HOLD arbitration scheme.

D.4.2 VL-Bus Clock

The VL-bus clock signal is a 1X clock that is in phase with the 486-type CPU and is driven by either the system logic or the local-bus controller. The VESA specification allows for a frequency range of up to 66 MHz and dynamic clock scaling. The specification limits the low-to-high level skew from the CPU clock to LCLK as shown in Table D–1.

Table D–1. VL-Bus Skew

LCLK Max Frequency	Unit	Max Skew	Unit
33	MHz	3	ns
40		2.5	
50		2	

Systems that currently support a 1X and a 2X clock source should supply the 2X clock source to the CLK2 input of the TI486SXL and the 1X clock source to the VL-bus LCLK signal.

Systems that currently support only a 2X clock source can consider the addition of a PLL or clock divider to generate the 1X VL-bus clock.

D.4.3 VL-Bus Slot ID Settings

The VL-bus slot ID settings are shown in Table D–2.

Table D–2. VL-Bus Slot ID Settings

Slot ID	Setting	Comments
ID0	1	TI486SXL Mode
ID1	0	TI486SXL Mode
ID2	0 or 1	0: Minimum one wait state for writes 1: Zero wait states for writes
ID3	0 or 1	0: >33 MHz CPU clock speed 1: < 33 MHz CPU clock speed
ID4	0	Burst transfer not supported

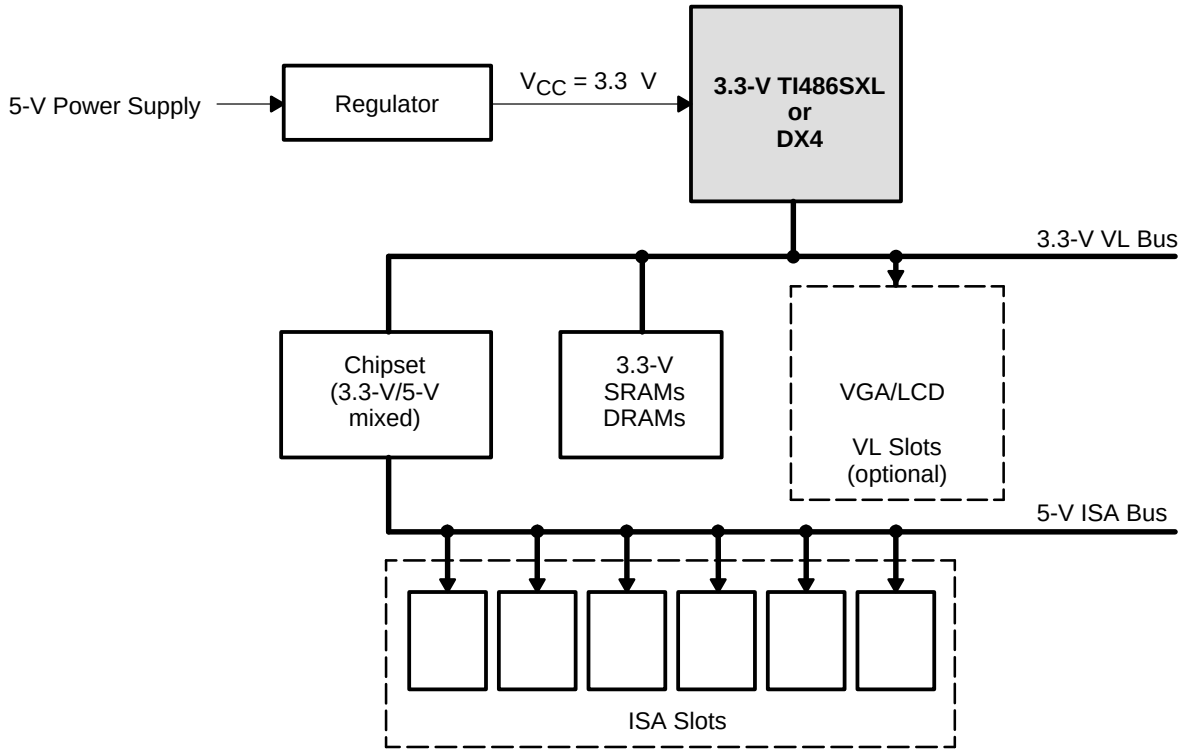
D.5 Power Planes for 3.3-V and 3.3-V/5-V Systems Using TI486SXL or 486DX4

Power planes for implementing 3.3-V-only and mixed 3.3-V/5-V systems, using the TI486SXL or the 486DX4, are described in the following subsections.

D.5.1 Power Planes for 3.3-V Systems

Figure D–4 shows the implementation of a 3.3-V system that supports use of either the TI486SXL or a 486DX4 microprocessor. This implementation yields a 5-V ISA bus and a 3.3-V VL bus with the microprocessor running at 3.3 V.

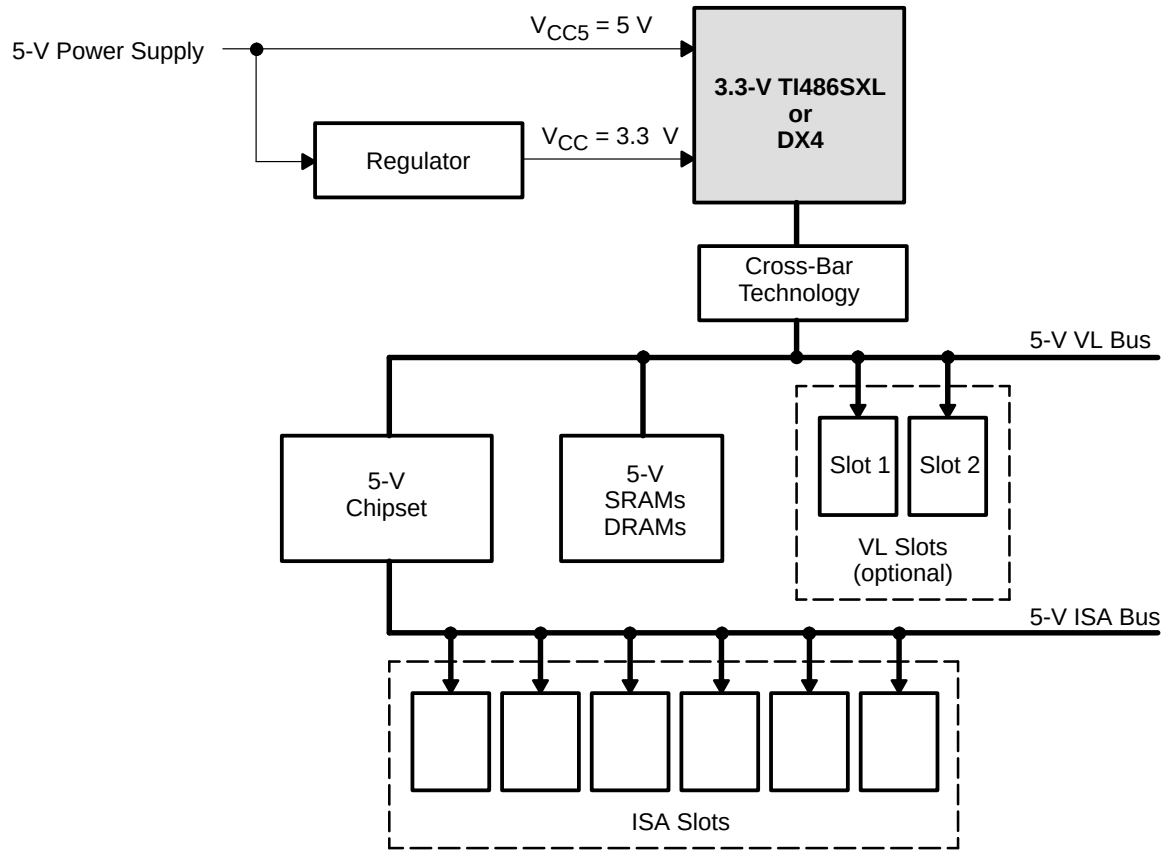
Figure D–4. 3.3-V VL-Bus Implementation



D.5.2 Power Planes for Mixed 3.3-V/5-V Systems

Figure D–5 shows the implementation of a 3.3-V/5-V system that supports use of either the TI486SXL or the 486DX4 microprocessor. This implementation yields a 5-V ISA and and a 5-V VL bus with the microprocessor running at 3.3 V.

Figure D–5. Mixed 3.3-V/5-V VL-Bus Implementation



D.6 Chipset Support

The following list of chipset vendors providing single-chipset solutions that support both the Intel/AMD and the TI486SXL interface was compiled from information received from the specified chipset vendors. This is a partial list and is not meant to be all inclusive.

- ACC Microelectronics
- Acer Laboratories
- EFAR
- ETEQ Microsystems
- Headland Technology
- OPTI
- PicoPower Technology
- SARC/PC Chip
- Silicon Integrated Systems (SIS)
- Symphony Laboratories
- Tidalwave
- UMC
- UniChip
- Western Digital

Appendix E

Thermal Management in Microprocessor-Based Systems

This appendix explains basic thermal concepts and the relationship between thermal measurements and the system. In addition, problems associated with comparing thermal specifications from different manufacturers are discussed. Finally, corrective activity within JEDEC is explained in detail. This information is intended for the casual scientific reader, and the only prerequisite is general engineering knowledge of semiconductor devices.

Topic **Page**

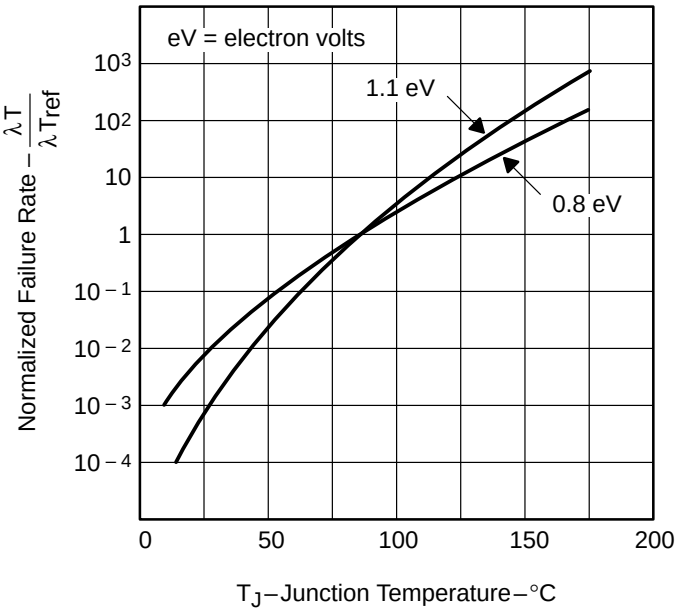
E.1 Introduction

Thermal management is considered to be an important factor in both the conception and usage of semiconductor integrated circuits (ICs). *Thermal management* is defined as the modes and techniques required to transfer a powered IC's resultant operating heat to a system thermal heat sink. The thermal management of an IC is normally discussed in terms of that IC's operating junction temperature (i.e., p-n junction of a diode). There are two main goals for thermal management.

- The first is to ensure that the operating junction temperature of the IC does not exceed the range of functional and maximum temperature limits of that IC. The *functional temperature range* of an IC is bounded by the temperatures that allow the IC to meet specified performance requirements. If the operating junction temperature of an IC is not within the functional temperature range, diminishing system performance and operational errors may result. The *absolute maximum temperature* is defined as the temperature at above which physical damage begins to occur to the IC.

- The second objective of thermal management is to ensure that the operating temperature of an IC enables the product reliability objectives for a given application to be met. Device failure rates are proportional to IC operating temperatures as shown in Figure E–1.

Figure E–1. Effect of Component Operating Temperature on Component Failure Rate†



† Richard C. Chu and Robert E. Simons, "Recent Developments For Electronic Packaging", Electronic Packaging Forum, Van Nostrand Reinhold, New York, 1991, pp. 183–189.

E.1.1 Thermal Impedance

Thermal impedance is an entity's resistance to heat dissipation through conduction, convection (natural and forced), and radiation. Thermal impedances are often analogous with electrical resistance, R, as described by Ohm's law in Equation E–1.

Equation E–1. Ohm's Law

$$R = \frac{V}{I}$$

where V represents voltage and I represents current. Similarly, thermal impedances (shown in the following equation), often denoted by R with a subscript of the Greek letter theta (Θ), can be described by the relationship in Equation E–2.

Equation E–2. Thermal Impedance

$$R_{\Theta} = \frac{\Delta T}{Q}$$

where ΔT represents the temperature difference between two reference points and Q is the heat-flow rate measured in watts. Heat-flow rate, Q, is often written as P or P_d.

E.1.2 Power

Power is defined as the rate of energy flow. This energy can be thought of as electrical energy or the resultant heat that is generated. Both electrical and heat energy are measured in watts. The power consumption of an integrated circuit is defined by Equation E–3:

Equation E–3. Power Consumption of an Integrated Circuit

$$P = V \cdot I$$

where V represents voltage and I represents current.

E.1.3 Junction Temperature

Indirectly, you can find the junction temperature (T_J) of a transistor or diode on a die using a temperature-sensitive electrical parameter (TSP) (see Figure E–2). Such a method is nondestructive and assumes that there is a uniform distribution of both current and temperature across the junction of the transistor or diode being used to conduct the test. Often the substrate diode (a diode used to reduce the amount of system noise) is used to conduct such a test. The diode's forward voltage drop is monitored while active and dissipating power as shown in Figure E–3. By controlling the temperature of a reference point and the voltage across the diode, you can find the junction temperature. This method of obtaining the junction temperature is precise and accepted throughout the semiconductor industry.‡

‡ Sherwin Rubin and Frank F. Oettinger, "Thermal Resistance Measurements on Power Transistors", *Semiconductor Measurement Technology: Thermal Resistance Measurements on Power Transistors*, US Government Printing Office, Washington, 1979, pp. 1–4.

Figure E–2. Die Using a Temperature-Sensitive Electrical Parameter

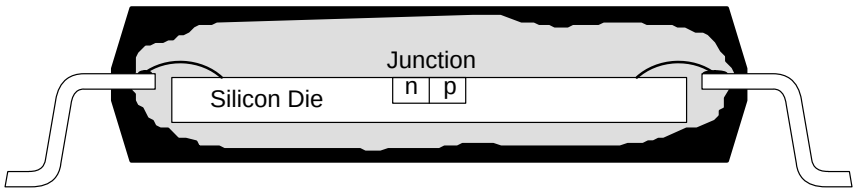
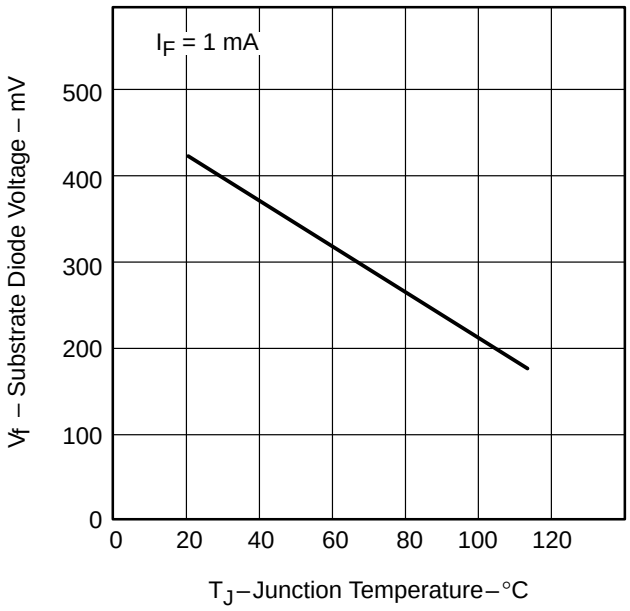


Figure E–3. Diode Voltage Versus Temperature for a Typical Bipolar Device



E.2 Modes of Heat Transfer

There are three ways that heat is transferred between points of differing thermal potential:

- Conduction
- Convection
- Radiation

Conduction, the simplest heat-transfer mechanism, is the transfer of kinetic energy via vibrations and collisions from a more excited atom or electron to a nearby atom or electron that is less excited. The ability to conduct heat is dependent on the material. For example, gases are poor heat conductors because of their low density. However, metals are good thermal conductors because their inherently high number of free electrons encourage collisions. This ability to conduct heat is quantified by a proportionality constant (k) often referred to as *thermal conductivity.* Table E–1 lists some common packaging materials and their associated thermal conductivities.

Table E–1. Thermal Conductivity of Packaging Materials[§,¶]

Metals (at 25°C)	Thermal Conductivity, (W/m) (°C)
Copper	397
Aluminum	238
Lead	34.7
Alloy-42 (common lead-frame material)	10.7

Gas (at 20°C)	Thermal Conductivity, (W/m) (°C)
Air	0.0234

Nonmetals	Thermal Conductivity, (W/m) (°C)
Glass	≈0.8
Epoxy glass	≈0.89

A second mode of heat transfer is *convection*, which is heat transfer by the movement of a heated substance. In the case of *natural convection,* such movement is caused by the induced differences in density that result from the expansion and compression of a gas or liquid subjected to temperature changes. Another type of convection, *forced convection*, forces movement of a cooling medium across a heat source. Often, forced convection is created by the use of a cooling fan within a system.

A final mode of heat transfer is *radiation*. Radiated heat transfers occur due to thermal emission primarily in the infrared spectrum. Radiation is subject to common-wave phenomena such as reflection. The ability of the surface of a material to radiate heat is referred to as that surface's *emissivity*. Possible values for emissivity are from zero to unity, where unity signifies the maximum thermal radiation at a given temperature.[§]

E.2.1 Integrated Circuit Thermal Resistance

The thermal resistance of an integrated circuit within a system can be broken into two major components:

- Internal resistance of the IC, $R_{\Theta i}$
- External resistance of the IC, $R_{\Theta x}$

Conventionally, resistances are discussed in more distinct terms. $R_{\Theta JC}$ is defined as the thermal impedance from the silicon die within an integrated circuit to the package surface or case of that IC. This thermal path includes the thermal impedance of each material used in packaging the IC, such as solder, die adhesive, base materials, leads, the case itself, etc. $R_{\Theta i}$ and $R_{\Theta JC}$ are interchangeable terms because $R_{\Theta JC}$ quantifies only those thermal impedances internal to the package ending at the package leads or package body surface. $R_{\Theta i}$ and $R_{\Theta JC}$ are functions of the IC package only and are not significantly affected by the particular system in which an IC is used. The semiconductor manufacturer controls the values of $R_{\Theta i}$ and $R_{\Theta JC}$.

¶ Charles A. Harper and Frank E. Altoz, *Electronic Packaging and Interconnection Handbook*, Mc Graw-Hill, Inc, New York, pp. 2.61–2.62.

§ Raymond A. Serway, *Physics for Scientists and Engineers*, Saunders College Publishing, Philadelphia, p. 545.

The thermal impedances that exist between the package case and the system ambient thermal sink are collectively defined as $R_{\Theta CA}$ (thermal impedance from case to ambient air). For a given package size and format, all such thermal impedances are primarily dependent on the particular system in which an IC is used (PWB thermal conductivity, presence of forced convection, etc.). These impedances are controlled by the user of the IC. Often $R_{\Theta JC}$ and $R_{\Theta CA}$ are referred to together as $R_{\Theta JA}$.

$R_{\Theta JA}$ can be qualitatively described as the thermal impedances between, and including, a silicon die heat source and the system ambient thermal heat sink.¶#

Table E–2 displays the relative sizes of $R_{\Theta JC}$ and $R_{\Theta CA}$. The table also displays the values of $R_{\Theta JC}$ and $R_{\Theta CA}$ as percentages of the corresponding value of $R_{\Theta JA}$ at 0 cubic feet per minute (cfm) airflow. All entries listed come from various data sheets of several manufacturers of 486-class microprocessors. Notice that $R_{\Theta JC}$ accounts for a maximum of 15 percent of $R_{\Theta JA}$. For all QFP packages listed, 9.6 percent of $R_{\Theta JA}$ is $R_{\Theta JC}$. For the PGA package, 15 percent of $R_{\Theta JA}$ is $R_{\Theta JC}$. As previously mentioned, the semiconductor manufacturer controls the value of $R_{\Theta JC}$ through various process parameters. Therefore, at maximum, $R_{\Theta JC}$ accounts for approximately 1/8th of the $R_{\Theta JA}$ value for packages listed. Stated differently, $R_{\Theta CA}$ (or the system), accounts for approximately 7/8ths of the total thermal resistance within a system.

Table E–2. Thermal Performance of Various 486-Class Microprocessors

Package	Material	Number of Pins	$R_{\Theta JC}$, (°C/W)	Percent of $R_{\Theta JA}$	$R_{\Theta CA}$, (°C/W)	Percent of $R_{\Theta JA}$
QFP	Metal	100	2	8.7	21	91.3
	Plastic	100	4	11.1	32	88.9
PGA	Ceramic	132	3	15.0	17	85.0

The system in which an integrated circuit is used is quite significant in the $R_{\Theta JA}$ value for that IC. As stated, at least 7/8ths of the thermal impedance from the silicon die to ambient air is due to the system. Significant effort is concentrated on thermally optimizing the system to improve the thermal performance of the ICs within.

Note that the IC user controls the design specifications for the majority of items that contribute to the quality of the thermal path. Several user-controlled system factors are available that can contribute to reducing the thermal resistance of an IC in the system design:

- PWB thermal conductivity
- Proximity/density of the ICs on a PWB
- Airflow

¶ Charles A. Harper and Frank E. Altoz, *Electronic Packaging and Interconnection Handbook*, Mc Graw-Hill, Inc, New York, pp. 2.61–2.62.

Jack Belani and B.J. Shanber, “Impact of Packaging Materials on Semiconductor Thermal Management”, Third Conference of Electronic Packaging: Materials and Processes & Corrosion in Microelectronics, Minneapolis, Minnesota, April 28–30, 1987, pp. 113–115, 118.

E.2.2 PWB Conductivity

The thermal conductivity of a PWB is determined by the individual thermal conductivities of materials that comprise the PWB. PWBs are nonhomogenous and normally consist of a base-laminate material, such as epoxy glass, and various amounts of other materials, such as traces or planes made of copper. The thermal conductivity varies little between commonly used PWB laminates. Since the thermal conductivities of commonly used routing metals are much higher than that of the PWB laminate (for example, 238 for aluminum and 397 for copper versus 0.89 for epoxy glass), the thermal conductivity of a PWB is proportional to the amount sof metal in the PWB.

Table E–3 shows the thermal conductivities of several PWBs made from FR-4, a type of epoxy glass. The boards vary by the number of signal layers and the number of ground layers (essentially, the copper volume). As copper volume increases from 0 to 6.9 percent, thermal conductivity increases by a factor of 90 or almost two orders of magnitude. This is a result of the higher thermal conductivity of copper compared to epoxy glass. The thermal conductivity of the PWB is proportional to the signal and ground metal content of a PWB. The area and thickness of metal on lower levels of a PWB, under the footprint of an IC, affects the thermal performance of that particular IC.||

Table E–3. Thermal Conductivity of PWBs With Various Amounts of Copper

Board Type and Layers	Signal-Layer Trace Width†	Ground-Layer Trace Width†	Copper Volume, (%)	Thermal Conductivity, (W/m) (°C)
FR-4	—	—	0	0.3
FR-4 2 layer	35 μm	—	1.0	3.7
FR-4 4 layer	35 μm	35 μm	3.5	13.6
FR-4 4 layer	35 μm	70 μm	6.9	26.9

† Trace thickness is 2 μm.

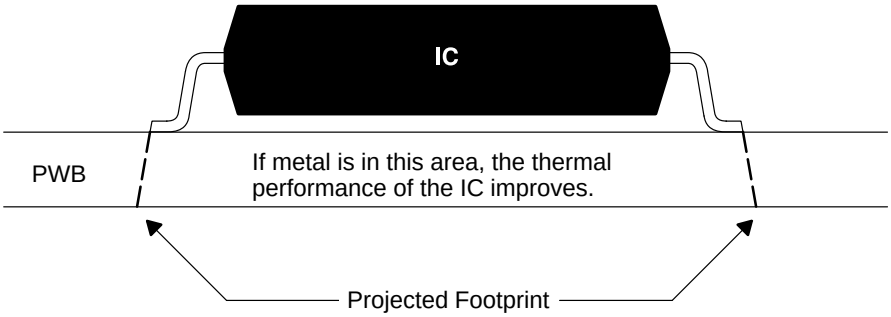
Table E–4 compares two types of PWBs with identical 100-pin QFP devices mounted on each board. Each board uses identical minimum-metal, signal-routing traces to complete the signal-interconnection layer. The single-sided PWB contains no metal on the opposite side of the board. The two-layer PWB has a solid-copper ground plane on the opposite side. All measurements are taken with no airflow present. The value of $R_{\Theta JA}$ for the IC is improved by 9 percent by the addition of a copper ground plane (a resulting increase of 55 percent in copper content). When metal is present on the lower levels of a PWB within the projected footprint of an IC (see Table E–4), the thermal performance of that IC is improved due to a lower value of $R_{\Theta JA}$.

|| Ake Malhammer, Ph.D, “Heat Dissipation Limits for Components Cooled by the PCB Surface”, International Electronics Packaging Conference, San Diego, California, September 15–18, 1991, pp. 307–308.

Table E–4. $R_{\Theta JA}$ Versus Board Type

Board Type	$R_{\Theta JA}$, (°C/W)
Single sided	36.0
Two sided	32.8

Figure E–4. Metal Within Projected Footprint of Integrated Circuit



E.2.3 Proximity of Integrated Circuit on Board

The location of an integrated circuit on a PWB can make a significant difference in the junction temperature of that device. In an ideal design, those ICs with the lowest heat dissipation are located in the center of a PWB, and those ICs with the highest heat dissipation are at the edges of the PWB. A concept known as the *territory surface method* associates an area of PWB required to sink the heat flow from a given IC. Often, in the case of surface-mount packaging, an IC's territory is violated by either other IC's territories or the edge of the PWB. In either case, thermal performance is hindered in all involved ICs. Note that not only the proximity of an IC on a PWB but also its relative location on the board has significant effects on thermal performance.[‖,☆]

E.2.4 Airflow

In a typical system, heat dissipated by natural convection is a significant portion of overall heat dissipation. When forced convection is present within a system, the amount of heat dissipation increases in proportion to the rate of flow of the convection. Higher rates of forced convection result in lower values of $R_{\Theta JA}$.

In Table E–5, values of $R_{\Theta JA}$ for varying amounts of forced convection are listed for a 100-pin QFP mounted on a single-sided board. As airflow (forced convection) increases from a rate of 0 cfm to 600 cfm, $R_{\Theta JA}$ is decreased by a factor of 2.4. As a rule, the $R_{\Theta JA}$ value of an IC in a system is inversely proportional to the presence/amount of forced convection (airflow).

‖ Ake Malhammer, Ph.D, "Heat Dissipation Limits for Components Cooled by the PCB Surface", International Electronics Packaging Conference, San Diego, California, September 15–18, 1991, pp. 307–308.

☆ M.M. Hussein, D.J. Nelson, and A. Elshahiu-Riad, "Thermal Interconnection of Semiconductor Devices on Copper-Clad Ceramic Substrates", 7th IEEE SEMI-THERM Symposium, August 1991, pp. 121–122.

Table E–5. $R_{\Theta JA}$ Versus Airflow

Airflow (cfm†)	$R_{\Theta JA}$
0	36
100	32
200	26
400	19
600	15

† cfm = cubic feet per minute

E.3 Thermal Specifications of Integrated Circuits

Manufacturers normally publish detailed specifications of ICs that contain a thermal portion, or a thermal specification. Manufacturer's thermal specifications differ in many ways, but most thermal specifications list the range of allowable package case temperatures (case temperatures at which the range of functional junction temperatures are not exceeded) to ensure that a device is functional. In addition, many manufacturers include some of the following variables:

- $R_{\Theta JC}$
- $R_{\Theta JA}$ at various airflows
- Maximum ambient air temperature, T_A, at various airflows.

Many thermal variables are system dependent. To compare ICs on the basis of their published thermal specifications, you must know the system in which such specifications were measured. Recall that 7/8ths of the thermal resistance of an IC, $R_{\Theta JA}$, is due to elements other than the IC. In addition, measurement techniques can affect thermal resistance values. In general, the following three factors prohibit comparisons between different manufacturers' thermal specifications:

- System dependence of $R_{\Theta JA}$ and $R_{\Theta CA}$
- Technique/location for measurement of T_A
- Definition of Q or P □

E.3.1 System Dependence of $R_{\Theta JA}$ and $R_{\Theta CA}$

There is presently no industry-accepted standard system for measuring thermal resistances. Consequently, systems used to measure thermal specifications vary widely between manufacturers with respect to thermal performance. For similar ICs built by different manufacturers, thermal specifications are often misleading due to different thermal systems.

□ [7]James A. Andrews, "Package Thermal Resistance Model Dependency on Equipment Design", IEEE Transactions on Components, Hybrids, and Manufacturing Technology, Volume II, Number 4, December 1988, pp. 536–537.

There are several approaches to publishing thermal specifications: worst case, best case, and somewhere between these two points. As a result, you need to be cautious when making decisions based on system-dependent thermal resistances such as $R_{\Theta JA}$ and $R_{\Theta CA}$. If information concerning the system is omitted from a thermal specification, values for $R_{\Theta JA}$ and $R_{\Theta CA}$ should be disregarded for the purpose of comparison.

E.3.2 Measurement of T_A

Recall Equation E–2 for thermal impedance, repeated here.

$$R_{\Theta} = \frac{\Delta T}{Q}$$

where ΔT is the difference in temperature between a transistor junction and some reference point.

The choice of reference point and its temperature with respect to the junction is of great importance to the precision of thermal impedance. Holding the junction temperature constant as the reference point's temperature is increased makes the calculated thermal impedance smaller. Most manufacturers choose the local ambient-air temperature within the system enclosure as the reference point. However, because the local air temperature is likely to be subject to natural convection and a resulting nonuniformity of temperature, the reference point must be well defined to avoid inaccuracy. If no information is included concerning the reference point, absolute comparisons of thermal specifications must be made cautiously.

E.3.3 Definition of Q

The rate at which electrical energy is converted into heat energy is known as power (P). P is defined by Equation E–4.

Equation E–4. Power

$$P = V_{CC} \times I_{CC}$$

For thermal-impedance calculations, some manufacturers use a relationship that describes the typical power dissipation as shown in Equation E–5.

Equation E–5. Power Dissipation

$$P = V_{CC(t)} \times I_{CC(t)}$$

where (t) denotes a typical value

Other manufacturers use the maximum amount of power dissipated as shown in Equation E–6.

Equation E–6. Maximum Power Dissipation

$$P = V_{CC(m)} \times I_{CC(m)}$$

where (m) is a maximum value.

Neither method is incorrect, but typical power dissipation is significantly lower than maximum power dissipation in most circumstances. As a result, thermal impedances calculated using typical power dissipation are lower than those thermal impedances calculated using maximum power dissipation. This is because of the inverse relationship between impedance and power dissipation (Q), as shown in Equation E–7.

Equation E–7. Relationship Between Impedance and Power Dissipation

$$R_{\Theta} = \frac{\Delta T}{Q}$$

When examining thermal specifications, note the manufacturer's definition of power dissipation. Often, the equation for power dissipation is included in either the publication section pertaining to electrical characteristics or the thermal specification's definition of variables. If not, use caution when comparing such specifications.

E.4 TI Thermal Specification Methodology

Some manufacturers publish thermal specifications according to the typical system conditions in which the IC is used. Some manufacturers publish thermal specifications for absolute worst-case conditions. Other manufacturers' thermal specifications are applicable for conditions somewhere between these two points. To ensure the reliability of Texas Instruments microprocessor devices, the thermal specifications are published in accordance with a realistic worst-case scenario. This means that the data is measured in a conservative manner, but not so conservative as to hinder its usefulness when designing microprocessor-based systems incorporating TI devices. The following paragraphs provide a detailed explanation of how TI obtains thermal data and the reasons for using such methods.

A thermal test die is mounted in the package to be tested and the package is mounted on a test board consisting of 0.062-inch thick FR-4 material with one-ounce copper etch. The 100-pin QFP (in this case, a package of the TI486SXLC microprocessor) is soldered to a single-sided test board using matching footprints and the minimum circuit-trace density required to interconnect the device electrically to the board. The 132-pin ceramic PGA (in this case, a package of the TI486SXL microprocessor) is inserted in a socket that is soldered to the same test board. As discussed previously, PWB thermal conductivity has a significant effect on the $R_{\Theta JA}$ value of a device and is proportional to the amount of metal in the PWB within the projected footprint of the device. Note that the test PWB described above has a minimum amount of routing metal and is a single layer. The PWB conductivity is minimized, and the experimentally determined value for $R_{\Theta JA}$ is maximized.

To measure still-air $R_{\Theta JA}$, the package to be tested and board on which it is mounted are placed horizontally in a container that has a volume of one cubic foot of air. Power is supplied to a transistor on the die, and after a thirty-minute stabilization period, the temperature of the air (T1) and the base-emitter volt-

age (VBE1) of the transistor are recorded. Power is supplied to an array of transistors on the die to cause an increase in junction temperature, and the base-emitter voltage (VBE2) of the powered transistor is recorded. The package and board are placed in an oven and the temperature is raised to 90°C (T2) and another measurement of base-emitter voltage (VBE3) is recorded. Still-air $R_{\Theta JA}$ can be calculated by substituting the measured variables (T1, T2, VBE1, and VBE3) into the following equations:

$$\text{slope} = \frac{(\text{VBE1}–\text{VBE3})}{(\text{T2}–\text{T1})}$$

$$R_{\Theta JA} = \frac{(\text{VBE1} - \text{VBE2})}{\text{slope}}$$

For the purpose of measuring $R_{\Theta JC}$, the package and board are placed in a bath of moving fluorinert FC-77. After a thirty-minute stabilization period, the temperature of the fluorinert is recorded (T1) and the voltage across a powered transistor on the test die is measured from base to emitter (VBE1). Power is then applied to an array of resistors on the test die to produce a subsequent increase in junction temperature. The voltage across the same transistor from base to emitter (VBE2) is recorded. The package and board are placed in an oven at 90°C (T2) and the voltage across the powered transistor is measured from base to emitter (VBE3). Note that at this point, the resistors are no longer powered. Once VBE1, VBE2, VBE3, T1, and T2 are known, these values are substituted into the last two equations to find a value for $R_{\Theta JC}$. $R_{\Theta JC}$ is independent of the system so system information has been omitted from this explanation. However, the test die that is used within the package must be consistent in size and power dissipation with the actual application die.

$$\text{slope} = \frac{(\text{VBE1}–\text{VBE3})}{(\text{T2}–\text{T1})}$$

$$R_{\Theta JC} = \frac{(\text{VBE1} - \text{VBE2})}{\text{slope}}$$

An example of plotted thermal data is shown in Figure E–5.

To measure $R_{\Theta JA}$ versus airflow, the test package and mounting board are placed vertically in a calibrated wind tunnel as shown in Figure E–6. A temperature probe and anemometer-type airflow probe are located towards the front end of the tunnel. A fan is mounted at the rear of the tunnel. Its airflow is directed away from the wind tunnel to induct air from the front of the tunnel to the rear. At various controlled rates of airflow, the voltage is measured across a powered transistor on the test die (VBE1). The temperature in the tube is recorded as T1. An array of resistors on the test die is powered to cause an increase of temperature across the die. The voltage is again measured across the same transistor (VBE2). The device is removed from the wind tunnel, placed in an oven at 90°C (T2), and only the transistor is powered. The voltage from base to emitter on the transistor is measured (VBE3). As in the procedure for $R_{\Theta JC}$, the experimental values are substituted into the equations for slope and $R_{\Theta JA}$ to find the value for the slope and $R_{\Theta JA}$ for a specific airflow.

Figure E–5. Plotted Die Thermal Data

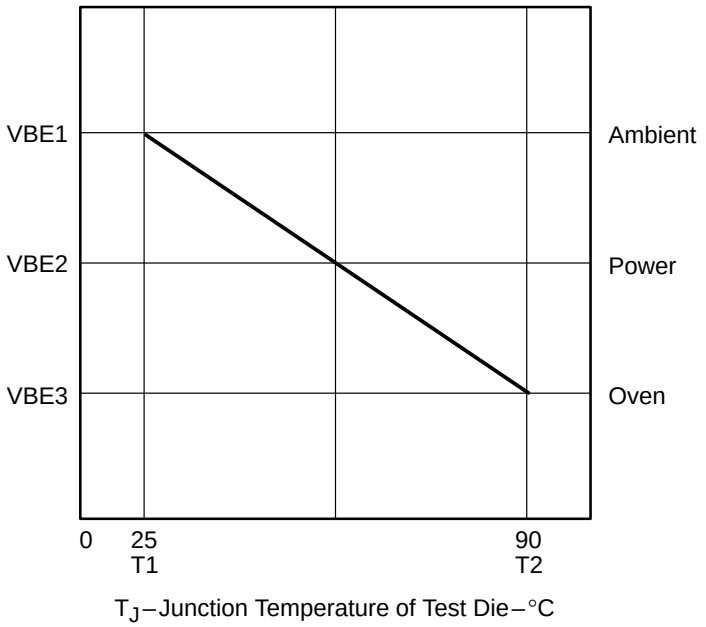
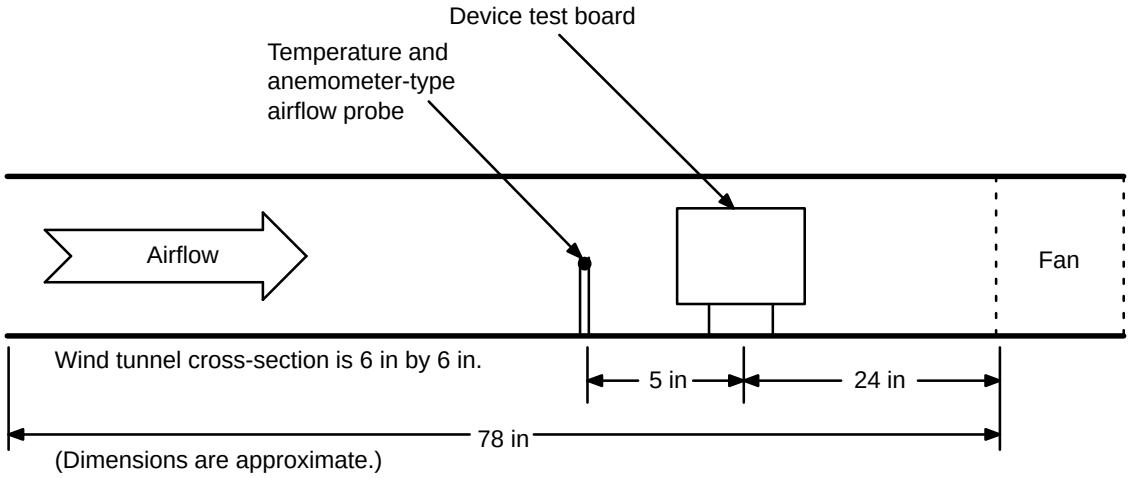


Figure E–6. Wind Tunnel Schematic



The procedures described above are relatively consistent across the industry with the exception of the test-board specifications and the measurement location of T1. In the test-board specification, the thermal conductivity is of great importance to the experimentally determined value of $R_{\Theta JA}$. As shown in Table E–3 (page E-7), a 4-layer FR-4 PWB is approximately 89 times as thermally conductive as a single layer PWB with no copper. It is not uncommon to find 8 or more layers in a microprocessor PWB. TI uses a single-sided test board with only one ounce of copper etch as opposed to a typical application multilayer PWB with a much higher content of copper etch, and consequently, better thermal conductivity.

The $R_{\Theta JA}$ values reported by TI should be viewed as worst-case versus typical for an application. The ambient temperature location is measured and is not affected by an increase in operational case temperature as would occur in a

typical closed-system-case application. Such a measurement of ambient temperature allows for a greater difference or delta between the junction temperature and the measurement reference point and, as a result, a higher value of $R_{\Theta JA}$. When comparing $R_{\Theta JA}$ values from Texas Instruments with other manufacturers, understand the test conditions of each manufacturer before drawing conclusions regarding which unit offers the best thermal performance.

E.5 Guidelines

The possibility of disparity in generating thermal specifications causes difficulty in comparisons of similar parts produced by different manufacturers. To ensure the validity of a comparison between the thermal specifications of several devices, follow these guidelines:

- Ensure that the system is the same for all devices included in the comparison. If the system is not the same, only consider values for $R_{\Theta JC}$. Disregard $R_{\Theta CA}$ and $R_{\Theta JA}$ values because of their system dependence.#
- Disregard from the comparison those devices whose thermal impedances were obtained using different reference points. Remember that ΔT decreases as the reference temperature increases (holding the junction temperature constant), and that thermal impedance is proportional to ΔT. An increase in ΔT (or a decrease in the measured reference temperature) causes a resulting increase in the calculated thermal impedance.
- Include only those devices with like definitions for power dissipation. Higher values for *P* result in lower values of calculated thermal impedance. Typical power dissipation (the product of typical V_{CC} and typical I_{CC}) is significantly lower than maximum power dissipation (the product of maximum V_{CC} and maximum I_{CC}).

E.6 Current Trends and Theory of Correction

The dilemma concerning thermal specifications and the incompatibilities between manufacturers has not gone unnoticed. The JEDEC JC-15 committee has developed objectives for standardizing electrical and thermal modeling and measurements for IC packages and interconnects. A task force, designated JC-15.1, was originated to accomplish two of the above goals by the following actions:

- Propose a standard board for device thermal-resistance measurements
- Provide a standard measurement to which actual thermal-modeling measurements can be compared

Jack Belani and B.J. Shanber, “Impact of Packaging Materials on Semiconductor Thermal Management”, Third Conference of Electronic Packaging: Materials and Processes & Corrosion in Microelectronics, Minneapolis, Minnesota, April 28–30, 1987, pp. 113–115, 118.

Companies often use varying systems and measuring techniques to obtain thermal-resistance measurements of ICs. To cope with these variances, JEDEC JC-15.1 proposes a board layout to standardize thermal-resistance measurements. The proposed board (3 in by 4.5 in) contains only the device to be characterized with a minimum amount of metal. If widely accepted within the semiconductor industry, such a board definition could improve the validity of comparisons between integrated-circuit thermal specifications.

E.7 Conclusions

In summary, the thermal impedance of an integrated circuit within a system is divided into two components:

- $R_{\Theta i}$ and $R_{\Theta x}$
- $R_{\Theta JC}$ and $R_{\Theta CA}$

$R_{\Theta JC}$ (or $R_{\Theta i}$) account for only about 1/8th of the total thermal resistance of an IC within a system. $R_{\Theta CA}$ (or $R_{\Theta x}$) is responsible for 7/8ths of the total thermal resistance. The total thermal resistance of an IC within a system, often referred to as $R_{\Theta JA}$, is significantly dependent on the system's thermal performance. The system thermal performance can be attributed to several factors:

- PWB thermal conductivity
- Proximity of ICs on the PWB and total component density of the PWB
- Presence/amount of forced convection

Thermal specifications of ICs include one or more of the following variables versus airflow: $R_{\Theta JC}$, $R_{\Theta JA}$, and $T_{A(m)}$. $R_{\Theta JA}$ is dependent on the system.

To make a valid comparison of multiple manufacturers' thermal specifications for similar parts, thermal specifications must meet the following guidelines:

- Identical systems (i.e., PWB thermal conductivity, airflow)
- Similar reference points for thermal-impedance calculation
- Like definitions of P

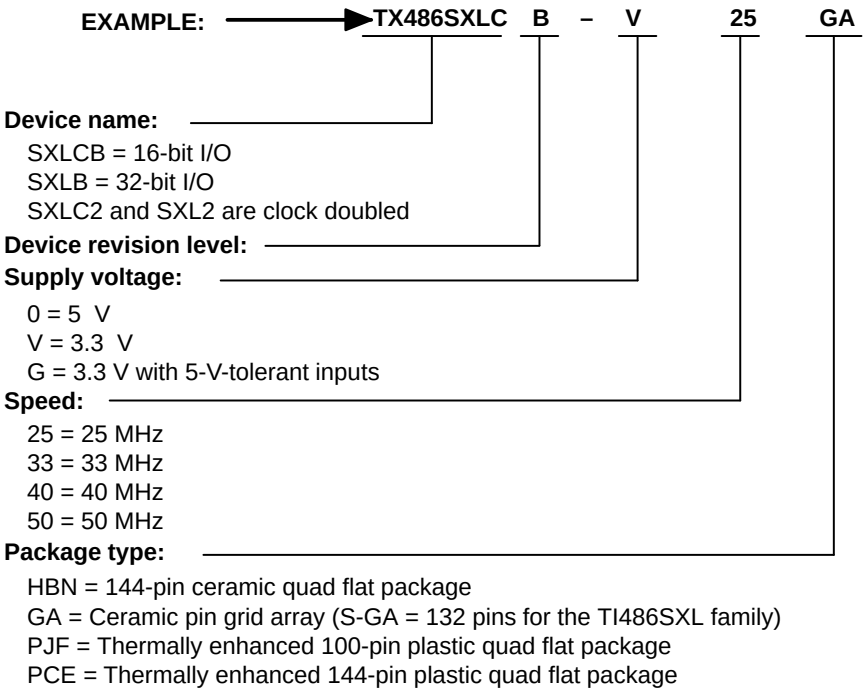
Because of the current problems surrounding thermal specification comparisons, JEDEC has provided a task force, JC-15.1, to develop and maintain a standard-PWB definition to measure thermal resistances to be included in thermal specifications. Until such a method is adopted industry wide, the discussed guildelines should be followed to assure valid thermal-specification comparisons.

Appendix F

Ordering Information

F.1 Part Number Components

Components of the TI486SXL(C) family of microprocessor part numbers are diagrammed in the following example.



F.2 Part Numbers for Microprocessors Offered

Table F–1 lists the complete part number for each version of the TI486SXL microprocessors offered. Table F–2 lists the part number for each version of the TI486SLC/DLC microprocessors offered. The tables provide a short description consisting of the supply voltage, performance capabilities, and the mechanical package for each device part number.

Table F–1. TI486SXLC and TI486SXL Part Numbers

Device Part Number	Supply Voltage (V)	Speed (MHz)		Package
		Core	Bus	
TX486SXLCB-V25-PJF	3.3	25	25	100-pin TEP plastic QFP
TX486SXLCB-040-PJF	5	40	40, 20†	
TX486SXLC2B-050-PJF	5	50	25	
TX486SXLB-040S-GA	5	40	40, 20†	132-pin PGA
TX486SXL2B-050S-GA	5	50	25	
TX486SXLB-040-PCE	5	40	40, 20†	144-pin TEP plastic QFP
TX486SXL-G40-HBN	3.3-V, 5-V tolerant	40	40, 20†	144-pin ceramic QFP
TX486SXL2-G50-HBN	3.3-V, 5-V tolerant	50	25	
TX486SXLB-040-HBN	5	40	40, 20†	
TX486SXL2B-050-HBN	5	50	25	
TX486SXL-G40-GA	3.3-V, 5-V tolerant	40	40, 20†	168-pin PGA
TX486SXL2-G50-GA	3.3-V, 5-V tolerant	50	25	
TX486SXLB-V40-GA	3.3	40	40, 20†	
TX486SXL2B-V50-GA	3.3	50	25	
TX486SXLB-040-GA	5	40	40, 20†	
TX486SXL2B-050-GA	5	50	25	

† These microprocessors can be operated as nonclock-doubled 40 MHz or clock-doubled 20/40 MHz.

Table F–2. TI486SLC/E and TI486DLC/E Part Numbers

Device Part Number	Supply Voltage (V)	Speed (MHz)	Package
TX486SLC/E-033C-PJF	5	33	100-pin TEP plastic QFP
TX486SLC/E-V25C-PJF	3.3	25	
TX486SLC/E-040C-PJF	5	40	
TX486DLC/E-033C-GA	5	33	132-pin ceramic PGA
TX486DLC/E-040C-GA	5	40	
TX486DLC/E-033C-PCE	5	33	144-pin TEP plastic QFP
TX486DLC/E-040C-PCE	5	40	

Appendix G

Glossary

2-way set associative: A type of cache in which an index identifies two lines of data (i.e., only two members of a set may exist in cache at a given time). This design provides significant performance improvement compared to direct-mapped caches as measured by the hit ratio. See *set associative*.

A

A20M: This pin is enabled when bit 2 of CCR0 is true. The A20M# pin is an anomaly occurring in PC designs as a result of the fact that truncated addresses can be generated by an 8086/8088 outside the physical address range.

A: *Accessed/nonAccessed* bit. Segment descriptor bit 8.

AC: *Alignment-Check*. An enable flag that verifies that computer-word bits are aligned with respect to significance.

address: A space in memory where a byte is assigned. Each byte of memory is assigned a specific address space. The amount of addressable memory space depends on the width of the CPU address bus. The TI486SXLC has a 24-bit address bus, and the TI486SXL has a 32-bit address bus.

AF: *Auxiliary carry Flag*. Set when an operation results in a carry out of (addition) or borrow into (subtraction) bit position 3. Otherwise, it is cleared.

AM: *Alignment-check Mask* bit. CR0 bit 18.

ARR1 through ARR4: *Address Region registers 1* through *4*. Define the location and size of the memory regions associated with the internal cache. These registers are unique to the TI486SXL(C) microprocessors.

assert: To apply a signal. A signal asserted is logically true.

AVL: *AVaiLable* bit. Segment Selector register bit 20.

B

bandwidth: The amount of information that can be transferred during a period of time. As an example, video, which requires a maximum bandwidth of 80 megabytes per second (MBps), takes advantage of the 132 MBps transfer rate provided by the VESA-VL or PCI bus.

BARB: A bit that, when bit 5 of CCR0 is set (high), enables flushing of the internal cache when a hold state is entered.

base : The beginning of some segments (extra data, code, or data segments) or the beginning address provided in some registers (CC3, GDTR, IDTR, or SD).

BIOS: *Basic Input Output System*. A set of routines that contain detailed instructions for activating computer and peripheral devices. The BIOS is normally implemented in nonvolatile memory.

bit: The fundamental unit of computer memory. A bit can be a 1 or a 0. A byte is made up of eight bits.

breakpoint: A point in a program at which to stop execution so that machine status may be determined.

byte: A sequence of eight bits. Represents one character of information.

C

cache: A small, high-speed memory used to provide a temporary storage location for data most likely to be requested by the CPU. This allows for quick access of data and improved CPU performance (i.e., zero wait states).

cacheable: The property of a memory location if the system allows data at this location to reside in the cache.

cache addressing: A type of addressing performed by dividing the physical address into an index field, a tag field, and a byte select field. A valid field indicates whether the cached data at that physical address is currently valid.

cache (data) coherency: A consistent relationship between data in cache memory and data in other memories. Data coherence is necessary when a system has multiple memories. If several memories contain the same data word, modifying that data word in one memory causes the data to be incoherent with the data stored in the other memories. Therefore, the other memories that have a copy of that same data word must either update or invalidate their copy.

cache flush: A memory operation used to maintain cache consistency. In a cache flush all locations with dirty bits are written to main memory. Then, the cache contents are cleared (flushed).

cache hit: Success by the CPU in retrieving requested data from cache memory.

cache miss: Failure by the CPU in retrieving requested data from cache memory because the data is not there.

cache tag address: Address that contains the high-order bits of the physical address of the associated data stored in the cache.

CCR0, CCR1: *Configuration Control Register 0 and Register 1.* Configuration control register 0 enables certain functions associated with cache control, suspend mode, and the clock-doubled mode. Configuration control register 1 sets up internal cache operation and system-management mode. These registers are unique to the TI486SXL(C) microprocessors.

CD: *Cache Disable* bit. CR0 bit 30.

CF: *Carry Flag*. Set when an operation results in a carry out of (addition) or borrow into (subtraction) the most significant bit. Otherwise, it is cleared.

CKD: (Enable) *Clock Doubled*. CCR0 bit 6.

clock doubled: A mode of microprocessor operation in which the internal core operates at the CLK2 frequency while the external bus interface operates at half the CLK2 frequency.

clock scaling: Changing the clock input frequency down or up. The TI486SXL(C) microprocessor family supports dynamic clock scaling.

clock speed: The speed at which the CPU operates, typically measured in megahertz (MHz).

CISC: *Complex Instruction Set Computer*. A type of computer architecture that requires multiple clock cycles per instruction but offers many specialized instructions for programmers.

conventional memory: DOS memory that occupies the addresses between 0 and 640K bytes and is available to the user or software programs.

coprocessor: An external processor that can be operated in parallel with the CPU to relieve the CPU load. The TI486SXL(C) microprocessors are designed to interface to a coprocessor.

CPGA: *Ceramic Pin Grid Array*. A package that consists of ceramic substrates to hermetically enclose the IC and an interconnection scheme that presents male leads extending from the bottom of the package.

CPL: *Current Privilege Level*. The privilege level of the current operation.

CPU: *Central Processing Unit*. The execution unit of the microprocessor. The CPU consists of control, shift, adder, multiplier, and limit units and a register file.

CR0, CR2, CR3: *Control registers 0, 2, and 3*. Control register 0 contains system control flags and indicates the general state of the CPU. The lower 16 bits are referred to as the machine status word. When paging is enabled and a page fault is generated, control register 2 retains the 32-bit linear address of the address that caused a fault. Control register 3 contains the 20-bit base address of the page directory.

CS: *Code Segment*. The register that holds a 16-bit segment base In real and virtual-8086 operating modes. In protected mode, the code segment register holds a segment selector.

D

D: *Default.* The default length bit for operands and addresses.

descriptor: A data structure that defines a segment's base, limit, and attributes.

DF: *Direction Flag*. A flag that, when cleared, causes string instructions to auto-increment (default) the appropriate Index registers (ESI and/or EDI). Setting DF causes auto-decrement of the Index registers.

direct-mapped cache: The simplest form of set associative cache architecture, one-way set associative. In a direct mapped cache, an index identifies only one line of data (i.e., only one member of a set may exist in cache at a given time). Therefore, only one address comparison is required to determine if the requested word is in the cache.

direct-memory access (DMA): Memory access that allows data to be transferred between a device and memory without the constant control of the CPU. DMA permits two operations to be executed simultaneously. As an example, the CPU can access the cache while DMA allows a peripheral to access the main memory.

disk drive controller: Electronic circuitry that transfers a copy of requested information or a software application from storage (disk drive, floppy drive, or CD-ROM) into RAM upon request by the microprocessor.

displacement: A value of up to 32 bits in length that is supplied as part of the instruction. The displacement is used as the address in direct address mode. The displacement is added to based, index, scaled index, based index with displacement, and based scaled index with displacement address modes.

DNA: *Device Not Available*. A fault indicating that the requested device is busy or missing.

DOS memory: Conventional memory that is limited to 1M byte of memory.

DP: *Displacement.*

DPL: *Descriptor Privilege Level* field. Gate or segment descriptor bits 14–13.

DRAM: *Dynamic Random Access Memory.* Volatile memory chips that use capacitors to store information as an electrical charge. DRAMs offer high density at a low cost, but must be refreshed frequently, which makes them relatively slow.

drive controller board: See *disk drive controller*.

DR0 through DR7: *Debug Registers 0* through *7.* Contain memory addresses and breakpoints used to support debugging of the microprocessor.

DS: *Data Segment*. In real and virtual-8086 operating modes, the data segment register holds a 16-bit segment base. In protected mode, the data segment register holds a segment selector.

DT: *Descriptor Type* bit. Segment selector register bit 12.

DTE: *Directory Table Entry.* Contains the starting address of the second-level page table. Selected from the directory table by the ten most-significant bits of the linear address.

DTI: *Directory Table Index.* Acts as a 32-bit master index to up to 1K individual second-level page tables.

E

E: Application descriptor bit. Segment descriptor bit 11.

EAX: *Extended* accumulator register.

EBP: *Extended* base pointer register.

EBX: *Extended* base register.

ECX: *Extended* count register.

EDI: *Extended* destination index register.

EDX: *Extended* data register.

EFLAGS: *Extended Flag* word register. Contains status information and controls certain operations on the microprocessor. The lower 16 bits of this register are referred to as the flag register.

EGA: *Enhanced Graphics Adapter*. A video standard for IBM-compatible PCs named after a particular video adapter that was the standard for the IBM PC-AT.

EIP: *Extended Instruction Pointe*r register. Contains the offset into the current code segment of the next instruction to be executed.

EM: *EMulate* processor extension. CR0 bit 2.

EPL: *Effective Privilege Level.* Protects memory from being accessed by privilege levels that are lower than the descriptor privilege level.

EPROM: *Electrically Programmable Read-Only Memory.* A permanent memory used for items such as the BIOS instructions, which occupy the reserved address space in DOS systems. EPROM access times tend to be long, but, being nonvolatile, they are used primarily for initialization. If higher performance is required, the EPROM contents can be copied to DRAM memory. This technique is called shadowing.

ES: *Extra Segment* register. The destination of STOS, MOVS, REP STOS, and REP MOVS instructions. Special segment override prefix ES allows the use of this additional segment register.

ESP: *Extended* General Purpose register.

expanded memory: A memory scheme that borrows addresses from reserved DOS memory to point to additional memory as a means of getting around the 1M byte DOS memory limit.

extended memory: A memory scheme used by some software applications to get around the 1M byte DOS memory limit.

F

far jump: A jump whose destination is in another code segment.

fast IDE: *Fast Integrated Device Electronics*. Provides data transfer of 16-bit wide data at speeds of up to 13 MBps.

flash memory: Cards designed for program storage, can be used in floppy and solid-state applications, and are ideal for applications that require frequent updates.

float: A condition during which all 3-state bidirectional and output terminals are placed in a high-impedance state to isolate the microprocessor from the system electrically.

flush: To invalidate the entire contents of cache memory.

footprint compatible: Compatible for installation in existing boards/systems.

FPU: *Floating Point Unit*. A part of the microprocessor that accelerates the computation of floating-point arithmetic. If a PC does not have an FPU, the CPU emulates floating-point instructions, which take more time to execute. See *CPU*.

FS: Additional data *Segment* register. Special segment override prefix of FS allows the use of this additional segment register.

fully associative: The most flexible type of cache placement policy. There is no single relationship between all of the addresses. The cache has to store the entire address of each block of words and compare its address with each of those in the cache until it finds a match.

G

G: Limit *Granularity* bit. Segment descriptor bit 23.

GD: *Global Disable*. Denies access to the debug register when set.

GDT: *Global Descriptor Table*. Part of the selector mechanism that contains segment descriptors used when the TI bit in the segment selector register is set to zero.

GDTR: *Global Descriptor Table Register*. A register that holds a 32-bit base address and 16-bit limit for the global-descriptor table.

graphic accelerators: A high-performance video display board for graphical user interface. Graphic accelerators have special circuitry which speeds up image processing. The CPU sends commands to the accelerator which executes them rather than having the CPU manipulating and sending data to the adapters. Objects are drawn on the screen rather than being transferred pixel by pixel. This reduces the amount of data that is transferred across the processor bus.

graphic adapter: A circuit board that translates the instructions from the CPU into information that the PC monitor can understand. Graphic adapters before and including VGA rely on the CPU to perform operations that manipulate the display image. Advanced adapters, which handle more data, have circuitry to speed up image processing directly on the graphic adapter card.

graphic coprocessor: A programmable chip that performs much of the processing required to display graphics on a video screen. A graphic coprocessor is fully programmable making it more flexible than a graphic adapter.

graphics mode: A video mode that divides images into thousands of dots, or pixels, to create text and detailed images.

GUI: *Graphical Users Interface.* A method of operating software applications that permits the user to interact with the computer by using icons and small graphics rather than by using text and commands.

green PC: An environmentally correct PC that reduces power consumption (by as much as 80% when compared to current models). The development of green PCs resulted from the Environmental Protection Agency's Star program.

GS: Additional data *Segment* register. Special segment override prefix of GS allows the use of this additional segment register.

H

hard drive controller: See *disk drive controller.*

hot insertion (or hot swapping): Plugging or unplugging PC cards without disrupting the host system's operation. Typically associated with PCMCIA.

I

IDE: See *Integrated Device Electronics.*

IDT: *Interrupt Descriptor Table.* An array of up to 256 8-byte interrupt descriptors, each of which points to an interrupt service routine.

IDTR: *Interrupt Descriptor Table Register.* A register that holds a 32-bit base address and 16-bit limit for the interrupt-descriptor table.

IF: *Interrupt Flag*. When set, maskable interrupts (INTR input pin) are acknowledged and serviced by the CPU.

index: A reference or initial value.

instruction: A machine-language command to the CPU. The TI486SXL(C) instructions are described in detail in Chapter 7, *Instruction Set*.

instruction set: A set of machine-language instructions that the architecture of the TI486SXL(C) CPU can execute.

integrated device electronics (IDE): An interface based on the ISA bus that uses the set of registers and commands originally used by the IBM PC-AT. This interface is the current favorite among most disk drive makers for hard disks because it is inexpensive and has a low command overhead. Drives using IDE interfaces integrate the controller and drive in one, making them more efficient than older drives. Therefore IDE drives and controllers do not need to translate commands from your microprocessor. IDE provides data transfer of 8-bit wide data at speeds of up to 5M byte ps. Fast IDE provides data transfer of 16-bit wide data at speeds of up to 13 MBps. See *fast IDE*.

INTR: *Interrupt*. A signal generated by external hardware that changes the normal sequential flow of a program by transferring program control to a selected service routine.

invisible: Contents (data, address components, and current states) of registers and stored data that the programmer cannot access, trap, or retrieve.

IOPL: *Input/Output Privilege Level*. Indicates the maximum current privilege level (CPL) permitted to execute I/O instructions. Also indicates the maximum CPL allowing alteration of the IF bit.

I/O: *Input/Output*

I/O bus (peripheral or system bus): The data path used to communicate with the various I/O or peripheral devices in the PC. Using this bus avoids loading down the time-critical local or processor bus with the I/O or peripheral devices.

I/O controller: Circuitry specialized for I/O operations. Most I/O devices have a controller that acts as its supervisor and interfaces with the CPU. The controller can be built either into the system board or on a separate adapter that is plugged into the system bus. Some controllers have their own special-purpose processors and some even have their own memory.

I/O device interface: An essential part of any PC that supports the communication between the CPU and the device or peripheral.

I/O mapped: Memory mapping in which I/O devices are mapped into the programmed I/O address space. Address decoding is easier since fewer address lines must be decoded.

K

KEN: The *KEN#* pin is enabled when bit 3 of CCR0 is true.

L

LDT: *Local Descriptor Table.* Part of the selector mechanism that contains segment descriptors used when the TI bit in the segment selector register is set to one.

LDTR: *Local Descriptor Table Register*. Holds a 16-bit selector for the local-descriptor table.

limit: Defines the maximum range.

line: The fixed unit of information transfer between cache and main memory.

line size: The amount of information in a line, which is defined as a number of bytes. Line size is one of the parameters that most strongly affects cache performance as it represents the amount of data the cache must retrieve during each cache line replacement (every cache miss).

linear address: In real mode, the physical address. The offset address is added to the product of the segment register multiplied by sixteen to produce the linear address.

In protected mode, the offset address is added to the base address to produce the linear address. If paging is disabled, the linear address is the physical address. If paging is enabled, the linear address is translated by the paging mechanism into the physical address.

local bus: The data path that connects peripherals directly to the CPU. The local bus is designed to transmit 32-bit data at the speed of a PC's processor. Two local bus standards are VESA-VL and PCI.

locality: Address memory in the neighborhood of locations recently accessed by programs.

LRU: *Least-Recently Used* bit. Indicates which of the cache two-way sets was more recently accessed.

M

math coprocessor: See *FPU*.

MBps: *Megabytes Per Second.*

Mbps: *Megabits Per Second.*

memory mapped: A type of memory addressing in which I/O devices can be mapped into physical memory addresses. Even though more addresses must be decoded with this interface, memory-mapped devices can be accessed using CPU instructions allowing for more efficient code. Memory mapping also offers more flexibility in protection than I/O mapping through memory management since a device can be inaccessible/fully accessible or visible but protected. Very few peripherals use memory-mapped ports except for video cards.

MMAC: *Main Memory Access*. Storing data in or retrieving data from main memory.

modem: A data communications device that translates (MOdulates) computer signals into tones and translates (DEModulates) tones back into computer signals to transfer data between computers over telephone lines.

monochrome: A video mode that uses only one color in varying intensities.

MP: *Monitor Processor* extension bit. CR0 bit 1.

multithreading: A software technique that allows an operating system or application to split tasks into subtasks, or threads, for improved speed and efficiency.

N

NC0: *Noncacheable 0*. Bit 0 in the configuration control register 0. When set, this bit sets the first 64K bytes at each 1M-byte boundary as noncacheable.

NC1: *Noncacheable 1*. Bit 1 in the configuration control register 0. When set, this bit sets sets 640K bytes to 1M-byte memory region as noncacheable.

NC (Terminal designator): *No external Connection*. Make none.

negated: Logically false, not true.

NMI: *Nonmaskable Interrupt*. A rising-edge-sensitive input that, when asserted, causes the processor to suspend execution of the current instruction stream and begin execution of an NMI service routine.

noncacheable memory: A memory system in which all shared memory locations are considered noncacheable. Access to the shared memory is never copied to the cache, and the cache never receives stale data.

nonvolatile memory: Memory in which the data content is maintained whether the power supply is connected or not. ROM and EPROM are examples of nonvolatile memory.

NT: *Nested Task* flag. A flag that, while executing in protected mode, indicates that the execution of the current task is nested within another task.

O

OA: *Offset Address*. A memory address that is the result of an offset calculation. Base address, index address, scale factor, and displacement are the components used, in various combinations, to calculate the offset address.

OF: *Overflow Flag*. A flag that is 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. The overflow flag is 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.

opcode: The physical implementation of an instruction in machine-readable code.

OS: *Operating System*. A master control program that supervises the functions and components of a computer system.

P

P: *Prefix*. A bit in a prefix byte. Also, *Present* bit (gate or segment descriptor bit 15).

paging: A memory management technique that provides direct access to small portions of stored data within a large segment of virtual memory space. Paging is useful in minimizing the amount of physical space required to service active routines.

parallel port: A communications port used mainly to send out data to be printed. A parallel port moves data in bytes (8-bits wide) or words (16-bits or 32-bits wide) depending on the application.

parity bit: The eighth bit or extra bit that is used to help detect errors.

PCD: *Page-level Cache Disable* bit. A bit located in test register 7. This bit corresponds to the PCD bit of a page-table entry.

PCI: *Peripheral Component Interconnect*. A board-level local-bus implementation for high-end PC applications. PCI is a fully independent bus that requires a PCI bridge to establish communication with the CPU bus. PCI is fully independent from the CPU and the CPU timing, and PCI can be used with non-X86 systems. PCI multiplexes addresses and data to reduce the number of required pins. Each card is uniquely identified by a special code that allows for autoconfiguration.

PCMCIA: *Personal Computer Memory Card International Association*. A peripheral bus standard that provides a way for the portable computer user to expand the memory, storage, communication, and other capabilities that are common to the desktop PC user. There are several types of PCMCIA cards: DRAM, flash memory, hard-disk drives, LANs, and modems. PCMCIA cards can be plugged into the PCMCIA expansion slot without opening the computer.

PDBR: *Page-Directory Base Register*. A register located in control register 3 that contains the 20-bit base address of the page directory.

PE: *Protected mode Enable* bit. CR0 bit 0.

peripheral bus: See *I/O bus*.

peripherals: Devices such as printers, fax machines, modems, and so forth that are external to the CPU.

peripheral interface: An essential interface of any PC that supports the communication between the CPU and the peripherals.

PF: *Parity Flag*. A flag set when the low-order 8 bits of the operation result contain an even number of ones. Otherwise the parity flag is cleared.

PFO: *Page-Frame Offset*. Part of the paging mechanism. The physical page frame data is selected by the first 12 bits of the linear address.

PG: *PaGing* enable bit. CR0 bit 31.

PGA: *Pin Grid Array*. A package that consists of substrates to hermetically enclose the IC and an interconnection scheme that presents male leads extending from the bottom of the package.

physical address: The 32-bit linear address when paging is disabled. When paging is enabled, the paging mechanism translates the linear address into a physical address. The physical address appears on the pins of the CPU.

pipelined addressing: A type of addressing that allows bus cycles to overlap, increasing the amount of time available for the memory or I/O device to respond. The NA# input to the CPU controls address pipelining.

pipelining: A series of suboperation stages, like fetching, decoding, execution, and address translation. Pipelining results in a continuous execution rate of one instruction per clock cycle.

pixel: The smallest information building block of an on-screen image. Screen resolution is usually expressed in the number of pixels making up the width and height of a complete on-screen image.

PL: *Privilege Level*. Implements a protection scheme. The values for privilege levels are 0 to 3. Level 0 is the most privileged and 3 the least privileged.

PLL: *Phase-Locked Loop*. In the TI486SXL(C) microprocessor, a PLL implements clock synchronization.

posted write: The process of buffering or storing address and data in a write buffer. In a write-through cache, read cycles are accelerated but write cycles are not. Through the use of a write buffer, write cycles can also be accelerated.

power management: A feature of some CPUs that shuts down unused parts of the computer to save power.

PQFP: *Plastic Quad Flat Package*. A package that consists of a metal substrate, IC, and interconnection scheme that presents leads extending from the four sides of the plastic encapsulated package. The leads are formed using a double break to create a planar foot on each lead that supports the package body above the seating plane. The thermally enhanced package includes a metal plate or slug near the mounting surface that enhances heat dissipation.

prefix: Bytes placed in front of an instruction to override segment defaults, change operand/address-size attributes, assert LOCK#, and repeat string instructions.

privilege level: In the protected mode, a designation that controls the use of privileged instructions, I/O instructions, and access to segments and segment descriptors.

protected mode: The microprocessor's operating mode when the PE bit of control register 0 is set. In protected mode, the enhanced memory management capabilities, which include segmentation and paging, are available. Code has one of four privilege levels, with some processor instructions restricted to the most-privileged code.

PTE: *Page Table Entry*. Selected from the page table by bits 21–12 of the linear address. The base address of the desired page frame.

PTI: *Page Table Index*. A 32-bit master index to up to 1K individual page frames.

PWT: *Page-level cache Write Through*. A bit in test register 7 that enables or disables the page-level cache function. This register bit corresponds to the PWT bit of a page-table entry.

Q

QFP: *Quad Flat Package*. A package that consists of a substrate, IC, and interconnection scheme encapsulated in plastic that presents leads extending from the four sides of the package. The leads are formed, using a double break, to create a planar foot on each lead that supports the package body above the seating plane.

R

R: Opcode or *register* bit.

R/W: *Read/Write* or *Readable/Writable* bit. Segment descriptor bit 9.

real memory: Memory that actually exists in the PC, or memory that is not borrowed from an external source.

real mode: The processing mode in which the microprocessor is backward compatible with 8086/8088 microprocessors. No hardware protection is provided for segment access or use, and there is no privileged code. TI486SXL(C) powers up or resets to real mode.

RF: *Resume Flag*. A flag used in conjunction with debug register breakpoints. The resume flag is checked at instruction boundaries before breakpoint exception processing. If set, any debug fault is ignored on the next instruction.

RISC: *Reduced Instruction Set Computer*. A type of computer that executes instructions in one clock cycle by limiting the number of instructions that are available.

ROM: *Read Only Memory*. A permanent, unchangeable memory used in the PC to accomplish system startup. ROM stores the BIOS programs needed to perform diagnostics and instruct the computer in various operations. When using DOS, the contents of the ROM are placed in reserved memory.

RPL: *Requested Privilege Level* field. Segment selector register bits 1–0.

S

scale factor: *Scale Factor*. A factor (1, 2, 4, or 8) by which the index address is multiplied when the offset mechanism calculates the offset address.

SCSI: *Small Computer System Interface*. A type of peripheral interface that offers hard disk data transfer rates of up to 10 MBps.

segmentation: A memory management technique that permits application-specific segmentation to improve the efficiency of memory space utilization.

serial port: A communication path based on a standard convention of transmitting two-way asynchronous serial data. A serial port moves data one bit at a time and can be half duplex (one direction at a time) or full duplex (both directions simultaneously).

serialization: Conversion of byte-wide data as input to serial bits in a stream as output.

set associative: A type of cache placement policy that has more than one set of direct-mapped caches operating in parallel. Several block locations are allowed for each cache index. The block can be placed in and retrieved from any set. This type of cache performs more efficiently than a direct mapped cache, but needs a wider tag field and additional logic to determine which set should receive the data.

SF: *Scale Factor* or *Sign Flag*. Set equal to the high-order bit of the operation result (0 indicates positive, 1 indicates negative).

shadowing: A technique used to improve system performance and achieve faster access by copying the contents of ROMs or EPROMs into DRAM.

s-i-b byte: A byte that includes the *s*, *Index*, and *Base* fields.

SIMM: *Single In-Line Memory Module*. A packaging technique for memory modules.

SM4: S*ystem Management* access region *4*. Sometimes called address region register 4. SMM memory space is defined by assigning address region 4 to SMM memory space.

SMAC: *System-Management Memory Access*. In normal mode, SMADS# address strobes are generated instead of ADS# for system-management memory accesses.

SMI: *System Management Interrupt*. An interrupt that causes the microprocessor to enter the system management mode, which allows various subsystems of the computer to be powered down under certain conditions. The system-management interrupt has a higher priority than any other interrupt, including NMI.

SMM: *System Management Mode*. A power management feature that allows various subsystems of the computer to be powered down when not in use to conserve power.

snooping: A method of maintaining cache consistency. The cache controller monitors the bus lines to detect any shared locations that are written to by another processor. When a common cache location is found, it is invalidated, and cache consistency is maintained.

SS: *Stack Segment*. A register that contains segment selectors that index into tables located in memory. These tables hold the base address for each segment as well as other information related to memory addressing.

SRAM: *Static Random Access Memory*. A high performance storage medium that does not require refresh.

SUS: *SUSpend*. A bit in configuration control register 0 that enables or disables the SUSP# and SUSPA# pins, which control entry into the suspend mode.

system bus: See *I/O bus*.

T

T: Opcode bit.

T1: The first clock of a nonpipelined bus cycle.

T1P: The first clock of a pipelined bus cycle.

T2: Subsequent clocks of a nonpipelined bus cycle. NA# has not been sampled asserted.

T2I: Subsequent clocks of a pipelined bus cycle. NA# has been sampled asserted, but there is not yet an internal bus request pending.

T2P: Subsequent clocks of a pipelined bus cycle. NA# has been sampled asserted, and there is an internal bus request pending.

tag: A directory that records what data is currently being stored in a cache.

TEP: *Thermally Enhanced Plastic*. A plastic package that includes a metal plate or slug near the mounting surface to enhance heat dissipation.

text mode: A video mode that divides the screen into character positions.

TF: *Trap enable Flag*. A flag that, when set, causes a single-step interrupt to occur after the next instruction completes execution. TF is cleared by the single-step interrupt.

Th: A hold acknowledge state.

TI: *Table Indicator* bit. Segment selector register bit 2.

Ti: A bus Idle state.

TLB: *Translation Look-Aside Buffer*. An on-chip, four-way, set-associative, 32-entry page-table cache. Contains the most recently accessed pages, which reduces the average time required to make virtual memory references.

TR: *Task register*. A register that holds a 16-bit selector for the current task-state segment (TSS) table. The TR is loaded and stored using the LTR and STR instructions, respectively.

TR3 through TR7: *Test registers 3* through *7*.

transfer rate: The rate at which data is moved from one component to another. Usually measured in megabits per second (Mbps) or megabytes per second (MBps). Some examples follow.

Component	Type	Transfer Rate
System bus	ISA EISA MCA	1 to 4 MBps 33 MBps 32 MBps
Local bus	32-bit VESA-VL PCI	132 MBps 132 MBps
Hard disk drives	IDE SCSI	4 MBps 5 MBps
Networks	Ethernet Token ring	10 MBps 4 to 16 MBps

TS: *Task Switched* bit. CR0 bit 3.

TSR: *Task State Register*. A register that is saved and restored using the SVTS and RSTC instructions, respectively.

TSS: *Task State Segment.* A table to which the processor saves the current CPU state during task switching before starting a new task.

type: Segment *Type* field. Gate or segment descriptor bits 11–8.

U

U/S: *User/Supervisor*. An attribute used in conjunction with the write/read attribute to implement protection at the page level. When set (user), the page is accessible at all privilege levels. When clear (supervisor), the page is accessible only when CPL ≤ 2.

V

V86: *Virtual 8086.* See *virtual-8086 mode.*

VESA: *Video Equipment Standards Association* VL-bus. A straight-forward expansion of the 486 host bus, meaning that it uses the 486 data, address, and control signals directly. A few more lines are added to allow bus mastering and other functions.

VGA: *Video Graphics Array*. The most popular color graphics system for IBM-compatible computers at the time of this writing.

virtual-8086 mode: The microprocessor operations when it handles segment loads as an 8086. The microprocessor switches to virtual-8086 mode when the virtual-8086 mode flag is set in protected mode.

virtual memory: Space temporarily borrowed from an external memory source, such as hard disk, and used to simulate a large amount of memory. Up to 64 terabytes of virtual memory can be addressed in 386- and 486-based systems.

visible: Contents (data, address components, and current states) of registers and stored data that the programmer can access, trap, or retrieve.

VM: *Virtual-8086 Mode* flag.

volatile memory: A memory in which the data content is lost when the power supply is disconnected.

VRAM: *Video Random Access Memories*. Memories used by designers of high-resolution graphics and imaging systems to enhance system performance and display more colors at higher resolutions.

W

wait state: The number of clock cycles the CPU has to wait for other operations to complete before continuing with its operations.

way: Defines the organization of a cache. A cache with a way 1 and a way 2 is a 2-way cache.

WP1, WP2, WP3: *Write Protected* access regions *1* through *3* bits. These bits, located in the configuration control register 1, define write protection and cacheability for three regions of memory space. The starting address and block size for each region is mapped in the address region registers 1 through 3.

WP: *Write Protect* bit. CR0 bit 16.

write back: An approach used to update the main memory. The CPU writes data into the cache and sets a dirty bit indicating that a word has been written into the cache but not into the main memory. The cache data is written back into the main memory at a later time and the dirty bit is cleared. Write-back accesses memory less than a write-through cache, but its cache control logic is more complex.

write protected: An attribute applied to segments to ensure that the requestor privilege level is sufficient to perform a write to that segment.

write-through cache: A type of cache used to update main memory. Data is written to the main memory while it is written to cache, or immediately afterwards. The main memory always contains valid data, and blocks in cache can be overwritten without data loss. The hardware implementation remains relatively simple.

Z

ZF: *Zero Flag*. A flag that is set if the operation result is zero. Otherwise the flag is cleared.

Index

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z