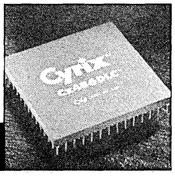
CYRIX Cx486DLC™ MICROPROCESSOR

Data Sheet

CYRIX Cx486DLC™ MICROPROCESSOR

High-Performance 486-Class CPU with Single-Cycle Execution and On-Chip Cache

Cyrix® Advancing the Standards



Introduction

♦ **DESIGNED FOR IBM-COMPATIBLE PERSONAL COMPUTERS**
- 486SX instruction set compatible
- Runs DOS, Windows and Unix
- 32-bit internal / 32-bit external data path
- 386DX bus compatible
- 40 MHz maximum clock frequency

♦ **GRAPHICS ACCELERATOR**
- On-board 16-bit hardware multiplier runs graphics faster than 386 and 486
- Ideal for Windows and other graphics-based applications (PC Mag Winmarks = $7x10^6$ pix/sec)

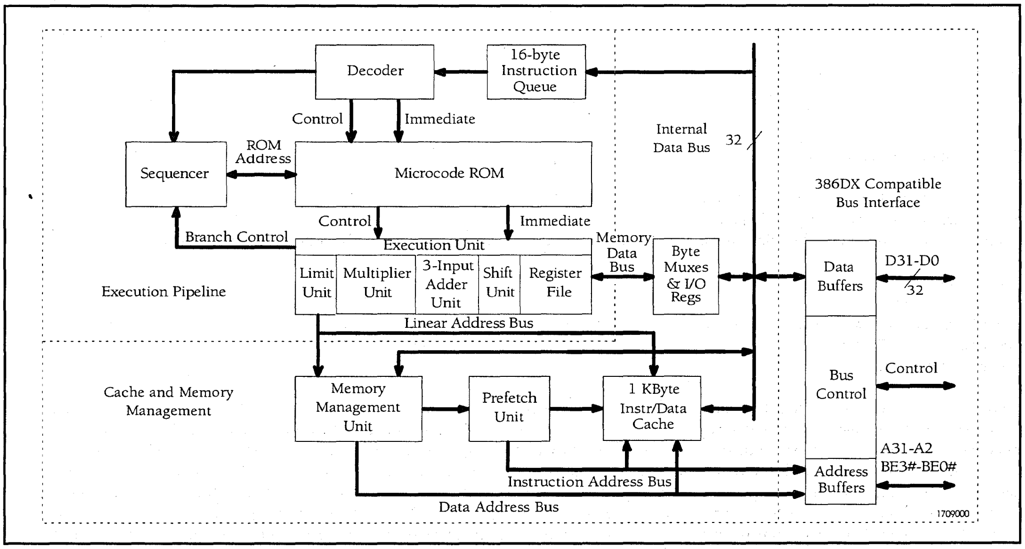
♦ **486-CLASS PERFORMANCE**
- Up to 2 times faster than 386DX at same clock frequency
- Landmark 2.0 = 130 MHz at 40 MHz
- Norton SI 6.0 = 66 at 40 MHz
- PM MIPS = 14 at 40MHz
- On-chip instruction and data cache
- High-speed single-cycle execution unit

♦ **LOW POWER CONSUMPTION**
- Software transparent suspend/resume
- Fully static design
- 0.10 mA I_{CC} at 0 MHz and 5 V

The Cyrix Cx486DLC is a high-performance microprocessor for use in IBM-compatible computers. The Cx486DLC executes the 486SX instruction set and all operating systems designed for this instruction set including DOS, Windows, and Unix.

The Cx486DLC includes a single cycle execution unit and a 32-bit internal data path that couple tightly to the on-chip 1 KByte cache. This enables the Cx486DLC to effectively access the cache two clocks faster than a zero wait-state external bus access. As a result, the Cx486DLC typically benchmarks 1.5 to 2 times faster than a 386DX at the same clock frequency.

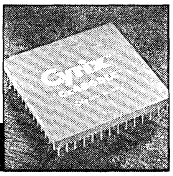
The Cx486DLC bus interface is compatible with existing 386DX hardware designs. Both hardware and software controls are provided by Cyrix to support the Cx486DLC cache interface and power management features allowing design flexibility with minimal changes to existing systems.



Order Number: 94076-01
Cyrix Corporation
2703 North Central Expressway
Richardson, Texas 75080
United States of America

CYRIX 486DLC™ MICROPROCESSOR

High-Performance 486-Class CPU with Single-Cycle Execution and On-Chip Cache



Advancing the Standards

TABLE OF CONTENTS

1

2

3

4

5

6

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

Cyrix® Advancing the Standards

1. PRODUCT OVERVIEW

1.1 Introduction

The Cyrix Cx486DLC microprocessor is an advanced 32-bit X86 compatible processor offering high performance and integrated power management on a single chip. The CPU is 486SX instruction set compatible and is backward compatible with the 386DX pinout. This CPU provides up to 2 times the performance of the 386DX at equal clock frequencies. The Cx486DLC is an ideal solution for battery-powered applications in that it typically draws 0.10 mA while the input clock is stopped in suspend mode.

The CPU supports 8, 16 and 32-bit data types and operates in real, virtual 8086 and protected modes. The Cx486DLC supports up to 4 GBytes of physical memory. This microprocessor 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 integrated instruction and data cache.

1.2 Execution Pipeline

The CPU execution path 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 which permit successive instruction execution overlap.

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 passed on to the microcode ROM stage of the pipeline. Non-sequential 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 non-sequential 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 actually completes the write cycle.

1.3 On-Chip Cache

The Cx486DLC on-chip cache maximizes overall performance by quickly supplying instructions and data to the internal execution pipeline. On 386DX systems, an external memory access takes a minimum of two clock cycles (zero wait states). For cache hits, the CPU eliminates these two clock cycles by overlapping cache accesses with normal execution pipeline activity.

The Cx486DLC cache is a 1 KByte write-through unified instruction and data cache. New cache lines are allocated only during memory read cycles. The cache can be configured as direct-mapped or as two-way set associative. The direct-mapped organization is a single set of 256 four-byte lines. When configured as two-way set associative, the cache organization consists of two sets of 128 four-byte lines and uses a Least Recently Used (LRU) replacement algorithm.

1.4 Power Management

The Cx486DLC power management features allow a dramatic reduction in current consumption when the CPU is in suspend mode (typically less than 2 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. The software initiates suspend mode through execution of the HALT instruction. Once in suspend mode, the CPU power consumption is further reduced by stopping the external clock input. The resulting current draw is typically 0.1 mA. Since the Cx486DLC is a static device, no internal CPU data is lost when the clock input is stopped.

1.5 Signal Summary

The Cx486DLC includes additions to the 386DX signal set which are shown in Figure 1-1. These additions consist of two power management signals (SUSP# and SUSPA#), four cache interface signals (FLUSH#, KEN#, RPLSET, and RPLVAL#), and an A20 mask input (A20M#). The signal set is described in greater detail in Chapter 3.

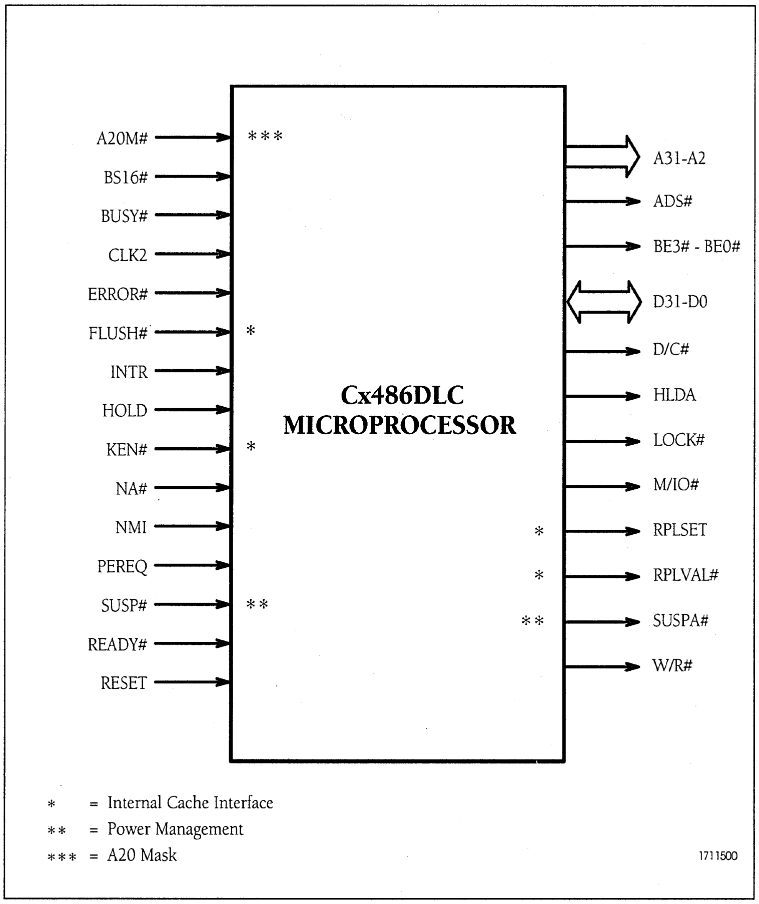


Figure 1-1. Cx486DLC Input and Output Signals

2. PROGRAMMING INTERFACE

In this chapter, the internal operations of the Cx486DLC are described mainly from an application programmer's point of view. Included in this chapter are descriptions of processor initialization, the register set, memory addressing, various types of interrupts and the shutdown and halt process. Also included is an overview of real, virtual 8086, and protected operating modes.

2.1 Processor Initialization

The Cx486DLC is initialized when the RESET signal is asserted. The processor is placed in real mode and the registers listed in Table 2-1 are set to their initialized values. RESET invalidates and disables the Cx486DLC cache, and turns off paging. When RESET is asserted, the Cx486DLC terminates all local bus activity and all internal execution. During the entire time that RESET is asserted, the internal pipeline is flushed and no instruction execution or bus activity occurs.

Approximately 350 to 450 CLK2 clock cycles (additional 2^{20} + 60 if self-test is requested) after deassertion of RESET, the processor begins executing instructions at the top of physical memory (address location FFFF FFF0h). Because paging is disabled, the linear address is the same as the physical address. When the first inter-segment JUMP or CALL is executed, address lines A31-A20 are driven low for code segment-relative memory access cycles. While A31-A20 are low, the Cx486DLC will execute instructions only in the lowest 1MByte of physical address space until system-specific initialization occurs via program execution.

Table 2-1. 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 0400 + Revision ID	Revision ID = 20h.
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 Register	Base = 0, Limit = 3FFh	
CR0	Machine Status Word	0000 0010h	
CCR0	Configuration Control 0	00h	
CCR1	Configuration Control 1	xxxx xxx0 (binary)	
NCR1	Non-Cacheable Region 1	000Fh	4-GByte non-cacheable region.
NCR2	Non-Cacheable Region 2	0000h	
NCR3	Non-Cacheable Region 3	0000h	
NCR4	Non-Cacheable Region 4	0000h	
DR7	Debug Register DR7	0000 0400h	

Note: x = undefined value

2.2 Instruction Set Overview

The Cx486DLC 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 Cx486DLC instructions operate on as few as 0 operands and as many as 3 operands. A NOP instruction (no operation) is an example of a 0 operand instruction. Two operand instructions allow the specification of an explicit source and destination pair as part of the instruction. These two operand instructions can be divided into eight groups according to operand types:

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

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

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

Chapter 6 of this manual lists each instruction in the Cx486DLC instruction set along with the associated opcodes, execution clock counts and effects on the FLAGS register.

2.2.1 Lock Prefix

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

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

An invalid opcode exception is generated if the LOCK prefix is used with any other instruction, or with the above instructions when no write operation to memory occurs (i. e., the destination is a register).

2.3 Register Set

There are 43 accessible registers in the Cx486DLC and these registers are grouped into two sets. The application register set contains the registers frequently used by application programmers, and the system register set contains the registers typically reserved for use by operating systems programmers.

The application register set is made up of:

Eight 32-bit general purpose registers
Six 16-bit segment registers
One 32-bit flag register
One 32-bit instruction pointer register

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

Three 32-bit control registers
Two 48-bit and two 16-bit system address registers
Six 32-bit debug registers
Two 8-bit and four 16-bit configuration registers
Five 32-bit test registers

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

2.3.1 Application Register Set

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

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

The **Segment Registers** contain segment selectors, which index into tables located in memory. These tables hold the base address for each segment, as well as other information related to memory addressing.

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

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

2.3.1.1 General Purpose Registers

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

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 lower16 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.

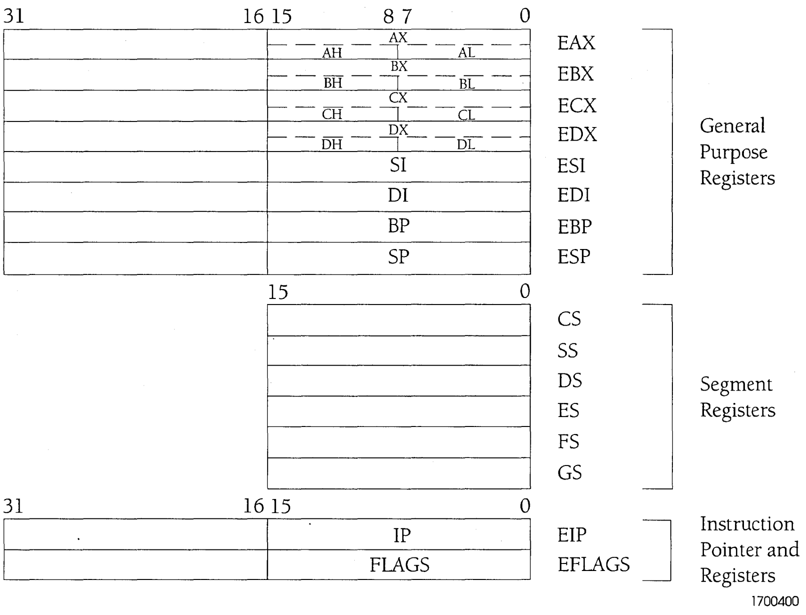


Figure 2-1. Application Register Set

Pointer and Index Registers

The pointer and index registers are listed below:

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

These registers can be addressed as 16- or 32-bit registers, with the "E" prefix indicating 32 bits. 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 Cx486DLC processor implements a stack using the ESP register. This stack is accessed during the PUSH and POP instructions, procedure calls, procedure returns, interrupts, exceptions, and interrupt/exception returns. The microprocessor automatically adjusts the value of the ESP during operation of these instructions. The EBP register may be used to reference data passed on the stack during procedure calls. Local data may also be placed on the stack and referenced relative to BP. This register provides a mechanism to access stack data in high-level languages.

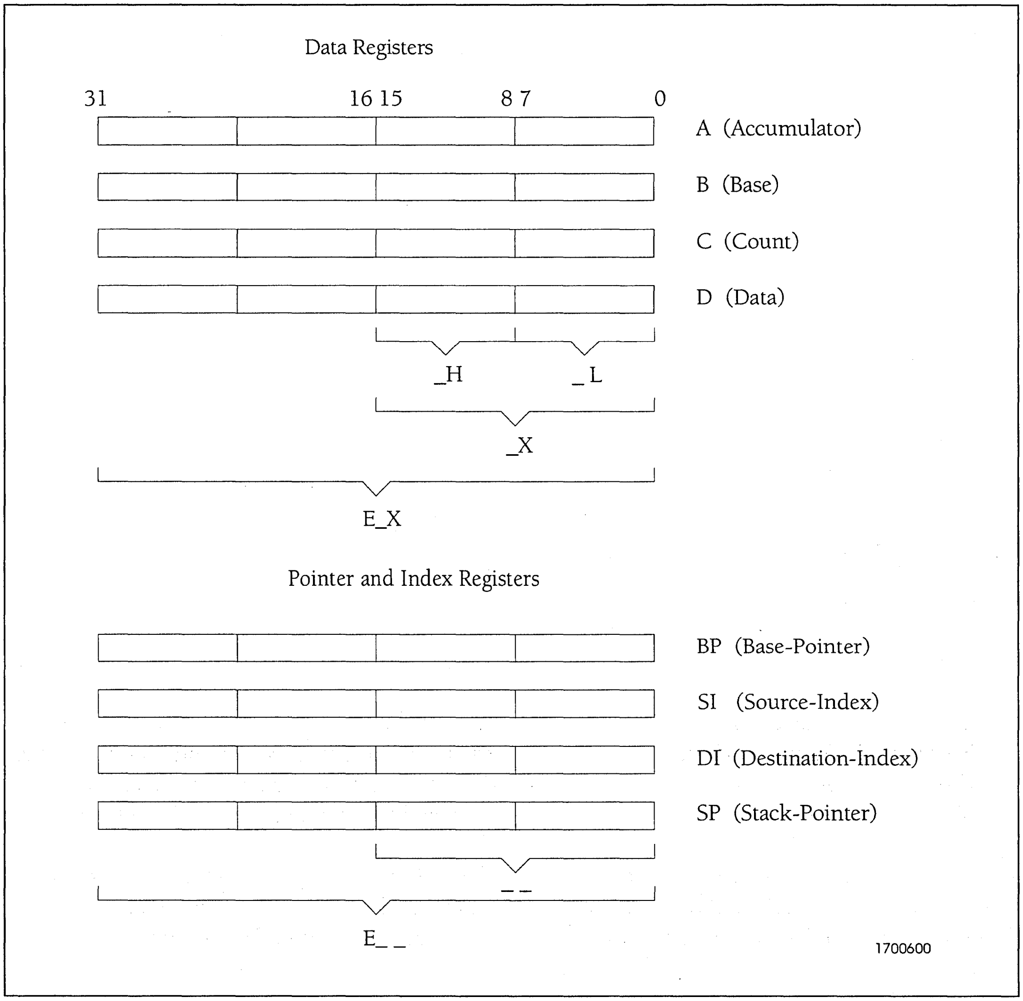


Figure 2-2. General Purpose Registers

2.3.1.2 Segment Registers and Selectors

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

There are six 16-bit segment registers:

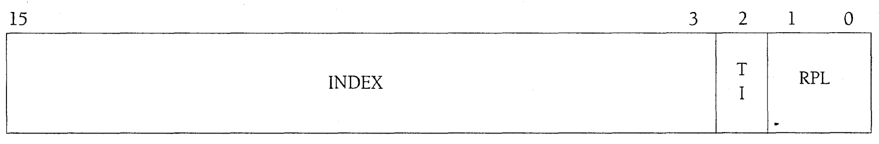
CS	Code Segment
DS	Data Segment
ES	Extra Segment
SS	Stack Segment
FS	Additional Data Segment
GS	Additional Data Segment

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

In protected mode, a segment register holds a **segment selector** containing a 13-bit index, a Table Indicator (TI) bit, and a two-bit requested privilege level (RPL) field as shown in Figure 2-3.

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

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



TI = Table Indicator
RPL = Requested Privilege Level

Figure 2-3. Segment Selector

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 and the EPL is the lesser 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 lesser privilege level.

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

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

Table 2-2. Segment Register Selection Rules

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

2.3.1.3 Instruction Pointer Register

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

2.3.1.4 Flags Register

The Flags Register, EFLAGS, contains status information and controls certain operations on the Cx486DLC microprocessor. The lower 16 bits of this register are referred to as the FLAGS register that is used when executing 8086 or 80286 code. The flag bits are shown in Figure 2-4 and defined in Table 2-3.

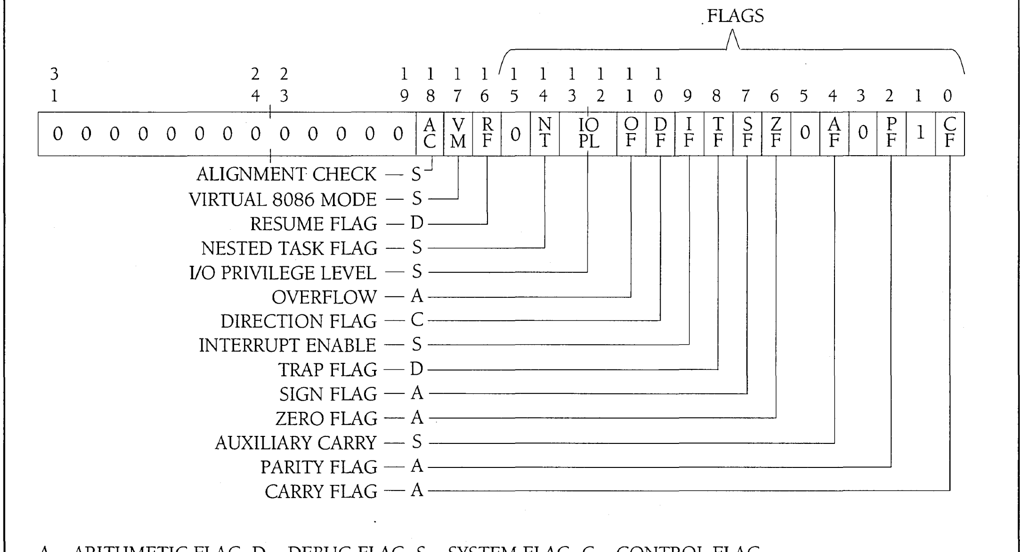


Figure 2-4. EFLAGS Register

Table 2-3. EFLAGS Bit Definitions

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

2.3.2 System Register Set

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

The **Control Registers** control certain aspects of the Cx486DLC microprocessor such as paging, coprocessor functions, and segment protection. When a paging exception occurs while paging is enabled, the control registers retain the linear address of the access that caused the exception.

The **Descriptor Table Registers** and the **Task Register** can also be referred to as system address or memory management registers. These registers consist of two 48-bit and two 16-bit registers. These registers specify the location of the data structures that control the segmentation used by the Cx486DLC microprocessor. Segmentation is one available method of memory management.

The **Configuration Registers** are used to control the Cx486DLC on-chip cache operation and power management features. The cache and power management features can be enabled or disabled by writing to these registers. Non-cacheable areas of physical memory are also defined through the use of these registers.

The **Debug Registers** provide debugging facilities for the Cx486DLC 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 1 KByte 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.

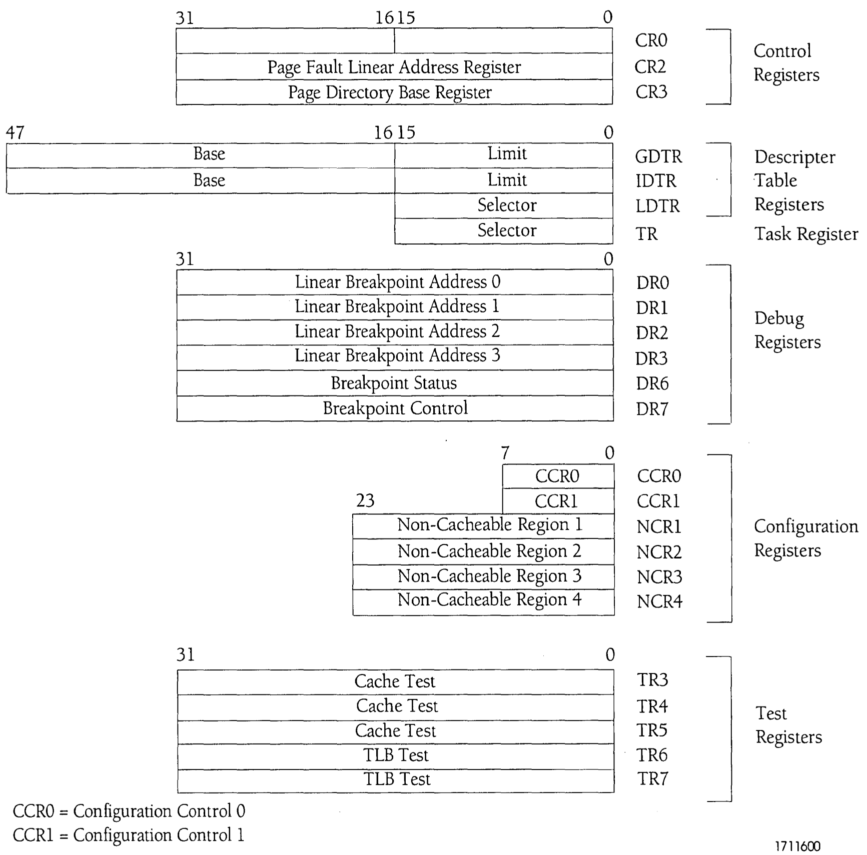


Figure 2-5. System Register Set

2.3.2.1 Control Registers

The control registers, CR0 through CR3, are shown in Figure 2-6. The CR0 register contains system control flags which control operating modes and indicate the general state of the CPU. The lower 16 bits of CR0 are referred to as the machine status word (MSW). The CR0 bit definitions are described in Table 2-4. The reserved bits in the CR0 should not be modified.

When paging is enabled and a page fault is generated, the CR2 register retains the 32-bit linear address of the address that caused the fault. CR3 contains the 20-bit base address of the page directory. The page directory must always be aligned to a 4 KByte page boundary, therefore, the lower 12 bits of CR3 should always be equal to zero.

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

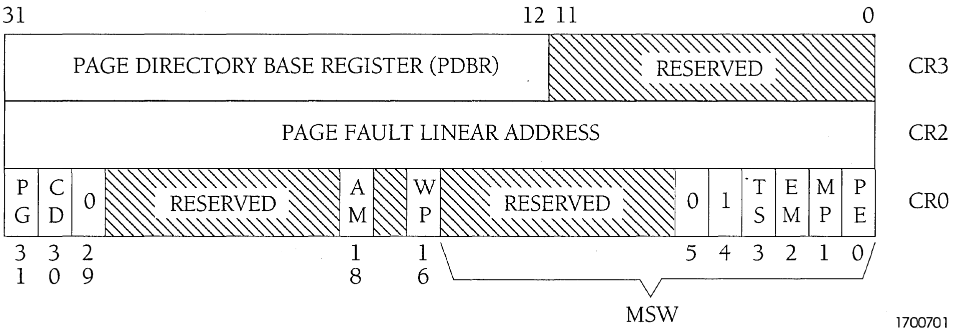


Figure 2-6. Control Registers

Table 2-4. 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, with segment based protection disabled, and addresses are formed as in an 8086-class CPU.
1	MP	Monitor Processor Extension: If MP=1 and TS=1, a WAIT instruction causes fault 7. The TS bit is set to 1 on task switches by the CPU. Floating point instructions are not affected by the state of the MP bit. The MP bit should be set to one during normal operations.
2	EM	Emulate Processor Extension: If EM=1, all floating point instructions cause a 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 attempt to modify.
5	0	Reserved: Do not attempt to 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 level 0-2. The Cx486DLC 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 from occurring. Reserved: Do not attempt to modify.
29	0	Cache Disable: If CD=1, no further cache fills occur. However, data already present in the
30	CD	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. Paging Enable Bit: If PG=1 and protected mode is enabled (PE=1), paging is enabled.
31	PG	

2.3.2.2 Descriptor Table Registers and Descriptors

Descriptor Table Registers

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

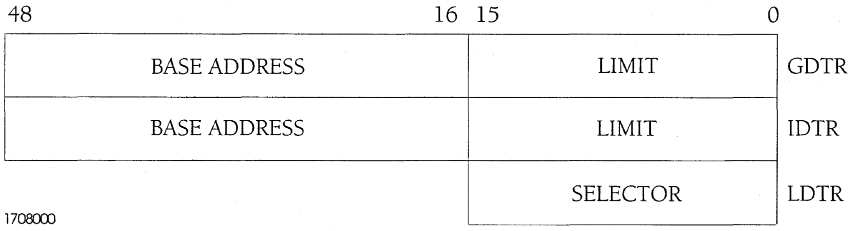


Figure 2-7. Descriptor Table Registers

The **Global Descriptor Table Register** (GDTR) holds a 32-bit base address and 16-bit limit for the Global Descriptor Table (GDT). The GDT is an array of up to 8192 8-byte descriptors. When a segment register is loaded from memory, the TI bit in the segment selector chooses either the GDT or the local descriptor table (LDT) to locate a descriptor. The index portion of the selector is used to locate a given descriptor within the descriptor 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 Cx486DLC 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 16-bit limit for the Interrupt Descriptor Table (IDT). The IDT is an array of 256 8-byte interrupt descriptors, each of which is used to point to an interrupt service routine. Every interrupt that may occur in the system must have an associated entry in the IDT. The contents of the IDTR are completely visible to the programmer.

The **Local Descriptor Table Register** (LDTR) holds a 16-bit selector for the Local Descriptor Table (LDT). The LDT is an array of up to 8192 8-byte descriptors. When the LDTR is loaded, the LDTR selector field 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 of isolating a given task's segments from other tasks in the system.

Descriptors

Descriptors are divided into three types: **Application Segment Descriptors** are used to define code, data and stack segments. **System Segment Descriptors** define an LDT segment or a TSS. **Gate Descriptor**s define task gates, interrupt gates, trap gates and call gates.

Application Segment Descriptors can be located in either the LDT or GDT. System Segment Descriptors can only be located in the GDT. Dependent on the gate type, gate descriptors may be located in either the GDT, LDT or IDT. Figure 2-8 illustrates the descriptor format for both Application Segment Descriptors and System Segment Descriptors and Table 2-5 lists the corresponding bit definitions.

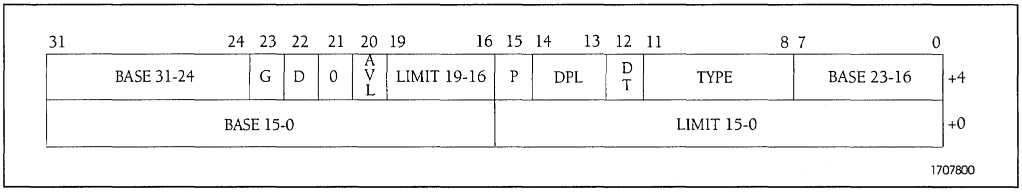


Figure 2-8. Application and System Segment Descriptors

Table 2-5. Segment Descriptor Bit Definitions

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

Gate Descriptors provide protection for executable segments operating at different privilege levels. Figure 2-9 illustrates the format for Gate Descriptors and Table 2-6 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.

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

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

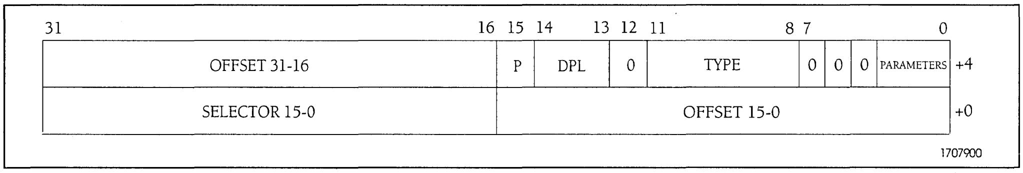


Figure 2-9. Gate Descriptor

Table 2-6. Gate Descriptor Bit Definitions

BIT POSITION	MEMORY OFFSET	NAME	DESCRIPTION
31-16 15-0	+4 +0	OFFSET	Offset used during a call gate to calculate the branch target.
31-16	+0	SELECTOR	Segment selector used during a call gate to calculate the branch target.
15	+4	P	Segment present.
14-13	+4	DPL	Descriptor privilege level.
11-8	+4	TYPE	Segment type: 0100 = 16-bit call 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.

2.3.2.3 Task Register

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

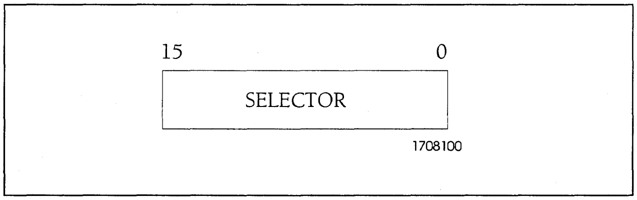


Figure 2-10. Task Register

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 286-type 16-bit TSS or a 386/486-type 32-bit TSS as shown in Figures 2-11 and 2-12. An I/O permission bit map is referenced in the 32-bit TSS by the I/O Map Base Address.

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.

1708200

Figure 2-11. 32-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	+16h
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 LEVEL 1	+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

1708800

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

2.3.2.4 Configuration Registers

The Cx486DLC provides six internal registers used to configure the internal cache and to enable or disable cache control and power management pins. These registers do not exist on any 80X86 microprocessors. Four of the registers are dedicated to defining non-cacheable areas of memory and the remaining two registers are used for Cx486DLC cache control and power management control as shown in Table 2-7.

Access to the Configuration Registers is achieved by writing the address (referred to as the index) of the register to I/O port 22h. I/O port 23h is then accessed to read or write data from or to the configuration register. Accesses to the on-chip configuration registers do not generate external I/O bus cycles. However, each I/O port 23h operation must be preceded by an I/O port 22h operation, otherwise the second and later I/O port 23h operations are directed off-chip and produce external I/O bus cycles. Accesses to I/O port 22h with an index outside of the C0-CFh range also result in external I/O cycles and do not affect the on-chip configuration registers.

Table 2-7. Configuration Registers Index Assignments

REGISTER NAME	REGISTER INDEX	NUMBER OF BITS IN REGISTER
CCR0 Configuration Control 0	C0h	8
CCR1 Configuration Control 1	C1h	8
Reserved	*C2h - C3h*	-
NCR1 Non-Cacheable Region 1	C4h - C6h	24
NCR2 Non-Cacheable Region 2	C7h - C9h	24
NCR3 Non-Cacheable Region 3	CAh - CCh	24
NCR4 Non-Cacheable Region 4	CDh - CFh	24
Reserved	*D0h - FFh*	-

Bit assignments for the configuration registers are listed in Table 2-8. The non-cacheable regions are defined by a starting address and a block size. The non-cacheable region block size ranges from 4 KByte to 4 GByte as shown in Table 9. The starting address of the non-cacheable region is restricted to block size boundary alignment. For example, a 128 KByte non-cacheable block is allowed to have a starting address of 0 KB, 128 KB, 256 KB, etc. This relationship between block size and starting address is true for all block sizes except 4 GBytes. When the block size is set to 4 GBytes, all physical memory is non-cacheable regardless of the setting of the starting address.

Table 2-8. Configuration Registers Bit Assignments

REGISTER NAME	REGISTER INDEX	BITS	DESCRIPTION
Configuration Control (CCR0)	C0h	0	NC0: If = 1, sets the first 64 KBytes at each 1 MByte boundary as non-cacheable.
		1	NC1: If = 1, sets 640 KBytes to 1 MByte region as non-cacheable.
		2	A20M: If = 1, enables A20M# input pin.
		3	KEN: If = 1, enables KEN# input pin.
		4	FLUSH: If = 1, enables FLUSH# input pin.
		5	BARB: If =1, enables flushing of internal cache when hold state is entered.
		6	CO: Selects cache organization: 0 = 2-way set associative 1 = direct-mapped
		7	SUSPEND: If = 1, enables SUSP# input and SUSPA# output pins. If = 0, output SUSPA# floats.
Configuration Control (CCR1)	C1h	0	RPL: If = 1, enables output pins RPLSET and RPLVAL#. If = 0, outputs RPLSET and RPLVAL# float.
		7 - 1	Reserved.
Non-Cacheable Region 1	C4h	7 - 0	Address bits A31 - A24 of Region 1 starting address.
	C5h	7 - 0	Address bits A23 - A16 of Region 1 starting address.
	C6h	7 - 4	Address bits A15 - A12 of Region 1 starting address.
		3 - 0	Region 1 Block Size (Table 2-8A).
Non-Cacheable Region 2	C7h	7 - 0	Address bits A31 - A24 of Region 2 starting address.
	C8h	7 - 0	Address bits A23 - A16 of Region 2 starting address.
	C9h	7 - 4	Address bits A15 - A12 of Region 2 starting address.
		3 - 0	Region 2 Block Size (Table 2-8A).
Non-Cacheable Region 3	CAh	7 - 0	Address bits A31 - A24 of Region 3 starting address.
	CBh	7 - 0	Address bits A23 - A16 of Region 3 starting address.
	CCh	7 - 4	Address bits A15 - A12 of Region 3 starting address.
		3 - 0	Region 3 Block Size (Table 2-8A).
Non-Cacheable Region 4	CDh	7 - 0	Address bits A31 - A24 of Region 4 starting address.
	CEh	7 - 0	Address bits A23 - A16 of Region 4 starting address.
	CFh	7 - 4	Address bits A15 - A12 of Region 4 starting address.
		3 - 0	Region 4 Block Size (Table 2-8A).

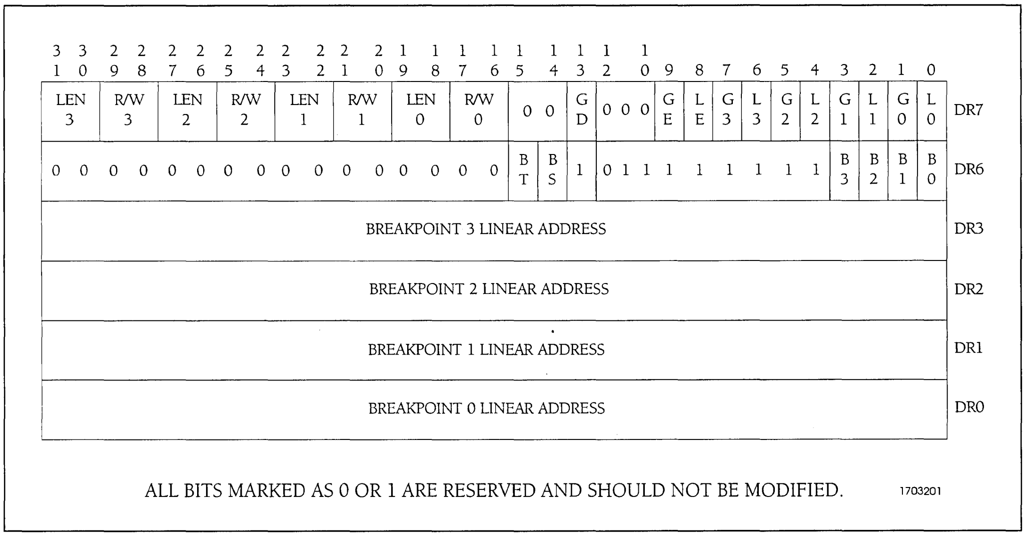
Note: All bits are cleared to 0 at reset, except C6h. C6h defaults to 0Fh to set the first non-cacheable region size = 4 GBytes.

Table 2-9. Non-Cacheable Regions Block Size Field

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

2.3.2.5 Debug Registers

Six debug registers (DR0-DR3, DR6 and DR7), shown in Figure 13, support debugging on the Cx486DLC. 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-byte, 2-bytes, or 4-bytes. The debug registers are accessed via MOV instructions which can be executed only at privilege level 0.



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 0 | 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 | 1 | 0 1 1 1 1 1 1 1 1 | B3 | B2 | B1 | B0 — DR6
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.
1703201

Figure 2-13. Debug Registers

The debug address registers DR0 - DR3 each contain the linear address for one of four possible breakpoints. Each breakpoint is further specified by bits in the debug control register (DR7). For each breakpoint address in DR0-DR3, there are corresponding fields L, R/W, and LEN in DR7 that specify the type of memory access associated with the breakpoint. The R/W field can be used to specify instruction execution as well as data access breakpoints. Instruction execution breakpoints are always taken before execution of the instruction that matches the breakpoint.

The debug status register (DR6) reflects conditions that were in effect at the time the debug exception occurred. The contents of the DR6 register are not automatically cleared by the processor after a debug exception occurs and, therefore, should be cleared by software at the appropriate time. Table 2-10 lists the bit definitions for the DR6 and DR7 registers.

Table 2-10. DR6 and DR7 Field Definitions

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

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

2.3.2.6 Test Registers

The five test registers, shown in Figure 2-14, are used in testing the CPU's translation look-aside buffer (TLB) and on-chip cache. TR6 and TR7 are used for TLB testing, and TR3-TR5 and used for cache testing.

TLB Test Registers

The Cx486DLC 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.

TR6 is the TLB Test Command 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.

TR7 is 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.

Tables 2-11 and 2-12 list the bit definitions for the TR6 and TR7 registers.

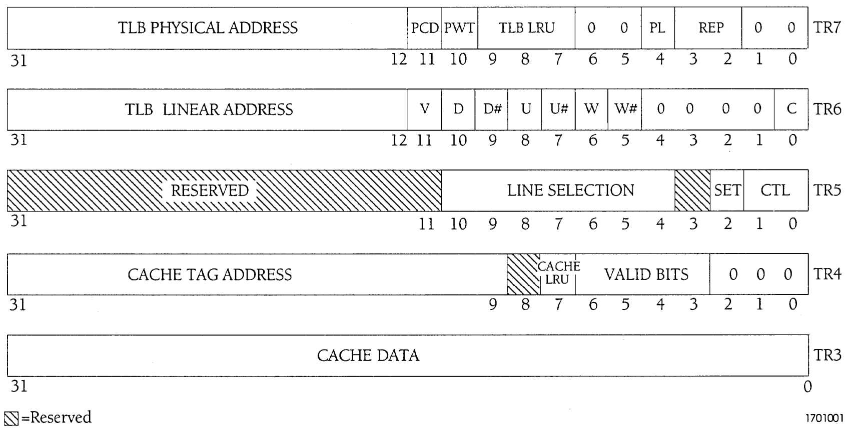


Figure 2-14. Test Registers

Table 2-11. TR6 and TR7 Bit Definitions

REGISTER NAME	BIT POSITION	DESCRIPTION
TR6	31-12	Linear address. TLB lookup: The TLB is interrogated per this address. If one and only one match occurs in the TLB, the rest of the fields in TR6 and TR7 are updated per the matching TLB entry. TLB write: A TLB entry is allocated to this linear address.
	11	Valid bit (V). TLB write: If set, indicates that the TLB entry contains valid data. If clear, target entry is invalidated.
	10-9	Dirty attribute bit and its complement (D, D#). (Refer to Table 2-11A).
	8-7	User/supervisor attribute bit and its complement (U, U#). (Refer to Table 2-11A).
	6-5	Read/write attribute bit and its complement (R, R#). (Refer to Table 2-11A).

	0	Command bit (C).
		If = 0: TLB write.
		If = 1: TLB lookup.

TR7	31-12	Physical address. TLB lookup: data field from the TLB. TLB write: data field written into the TLB.
	11	Page-level cache disable bit (PCD). Corresponds to the PCD bit of a page table entry.
	10	Page-level cache write-through bit (PWT). Corresponds to the PWT bit of a page table entry.
	9-7	LRU bits. TLB lookup: LRU bits associated with the TLB entry prior to the TLB lookup. TLB write: ignored.
	4	PL bit. TLB lookup: If = 1, read hit occurred. If = 0, read miss occurred. TLB write: If = 1, REP field is used to select the set. If = 0, the pseudo-LRU replacement algorithm is used to select the set.
	3-2	Set selection (REP). TLB lookup: If PL = 1, set in which the tag was found. If PL = 0, undefined data. TLB write: If PL = 1, selects one of the four sets for replacement. If PL=0, ignored.

Table 2-11A. 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 is the bit is 1 or 0.	Undefined.

Cache Test Registers

The Cx486DLC on-chip cache is a two-way set associative memory with 128 entries per set. Each TLB entry consists of a 23-bit tag, 32-bit data, and four valid bits. The 23-bit tag represents the high-order 23 bits of the physical address. The 32-bit data represents the four bytes of data currently in memory at the physical address represented by the tag. The four valid bits indicate which of the four data bytes actually contain valid data.

The Cx486DLC contains three test registers that allow testing of its internal cache. Using these registers, cache test writes and reads may be performed. Cache test reads allow inspection of the data, valid bits and the LRU bit for the cache entry. Cache test writes cause the data in TR3 to be written to the selected set and line in the cache. For data to be written to the allocated line, the valid bits for the line must be set prior to the write of the data. Bit definitions for the cache test registers are shown in Table 2-12.

Table 2-12. TR3-TR5 Bit Definitions

REGISTER NAME	BIT POSITION	DESCRIPTION
TR3	31-10	Cache data. Cache read: data accessed from the cache. Cache write: to be written into the cache.
TR4	31-9	Tag address. Cache read: tag address from which data is read. Cache write: data written into the tag address of the selected line.
	7	LRU. Cache read: the LRU bit associated with the cache line. Cache write: ignored.
	6-3	Valid bits. Cache reads: four valid bits for the accessed line (one bit per byte). Cache writes: valid bits written into the line.
TR5	10-4	Line Selection. Selects one of 128 lines.

	2	Set selection.
		If = 0: set 0 is selected.
		If = 1: set 1 is selected.

	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 flush (marks all entries as invalid).

2.4 Address Spaces

The Cx486DLC can directly address either memory or I/O space. Figure 2-15 illustrates the range of addresses available for memory address space and I/O address space. For the Cx486DLC, the addresses for physical memory range between 0000 0000h and FFFF FFFFh (4 GBytes). The accessible I/O addresses space ranges between 0000 0000h and 0000 FFFFh (64 KBytes). The coprocessor communication space exists in 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 is executed. The I/O locations 22h and 23h are used for Cx486DLC configuration register access.

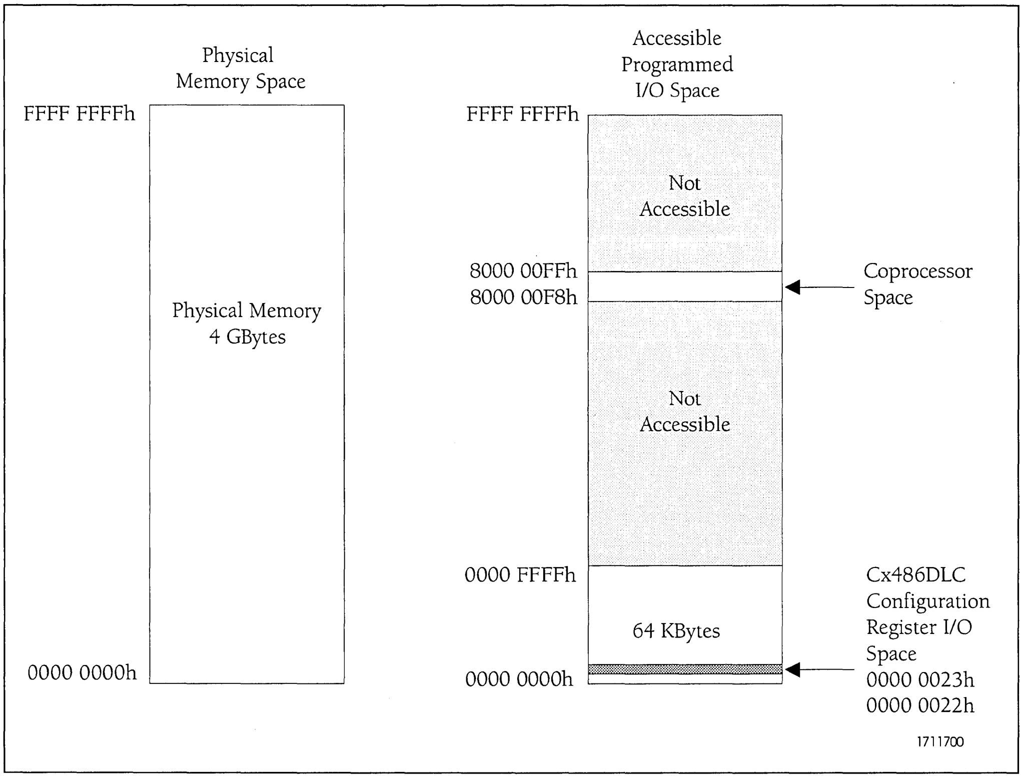


Figure 2-15. Memory and I/O Address Spaces

2.4.1 I/O Address Space

The Cx486DLC I/O address space is accessed using IN and OUT instructions to addresses referred to as "ports". The accessible I/O address space is 64 KBytes and can be accessed as 8-bit, 16-bit or 32-bit ports. The execution of any IN or OUT instruction causes the M/IO# pin to be driven low, thereby selecting the I/O space instead of memory space for loading or storing data. The upper 8 address bits are always driven low during IN and OUT instruction port accesses.

The Cx486DLC 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 Cx486DLC index range (C0h to CFh) result in external I/O cycles and do not effect the on-chip configuration registers. Reads of port 22h are always directed off-chip.

2.4.2 Memory Address Space

The Cx486DLC directly addresses up to 4 GBytes of physical memory. Memory address space is accessed as bytes, words (16-bits) or doublewords (32-bits). Words and doublewords are stored in consecutive memory bytes with the low-order byte located in the lowest address. The physical address of a word or doubleword is the byte address of the low-order byte.
With the Cx486DLC, memory can be addressed using nine different addressing modes. These addressing modes are used to calculate an offset address often referred to as an effective address. Depending on the operating mode of the CPU, the offset is then combined using memory management mechanisms to create a physical address that actually addresses the physical memory devices.

Memory management mechanisms on the Cx486DLC 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 Cx486DLC memory address space.

2.4.2.1 Offset Mechanism

The offset mechanism computes an offset (effective) address by adding together up to three values: a base, an index and a displacement. The base, if present, is the value in one of eight 32-bit general registers at the time of the execution of the instruction. The index, like the base, is a value that is 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 added to 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 2-16 illustrates the calculation of the offset address.

Nine valid combinations of the base, index, scale factor and displacement can be used with the Cx486DLC instruction set. These combinations are listed in Table 2-13. The base and index both refer to contents of a register as indicated by [Base] and [Index].

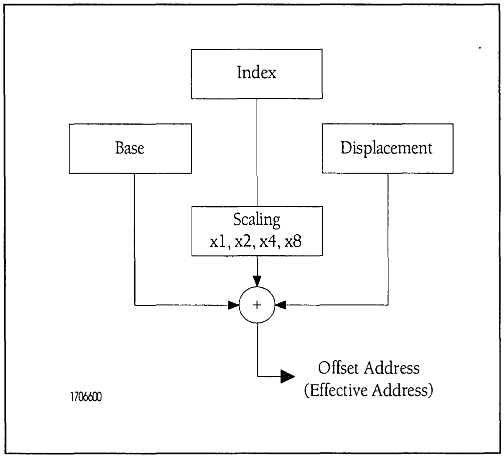


Figure 2-16. Offset Address Calculation

Table 2-13. Memory Addressing Modes

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

2.4.2.2 Real Mode Memory Addressing

In real mode operation, the Cx486DLC only addresses the lowest 1 MByte (2^{20}) of memory. To calculate a physical memory address, the 16-bit segment base address located in the selected segment register is multiplied by 16 and then the 16-bit offset address is added. The resulting 20-bit address is then extended with 12 zeros in the upper address bits to create the 32-bit physical address. Figure 2-17 illustrates the real mode address calculation. Physical addresses beyond 1 MByte cause a segment limit overrun exception.

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

2.4.2.3 Protected Mode Memory Addressing

In protected mode three mechanisms calculate a physical memory address (Figure 2-18).

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

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

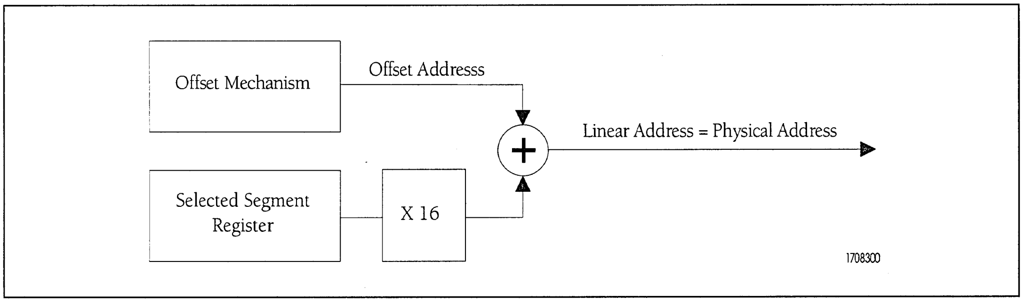


Figure 2-17. Real Mode Address Calculation

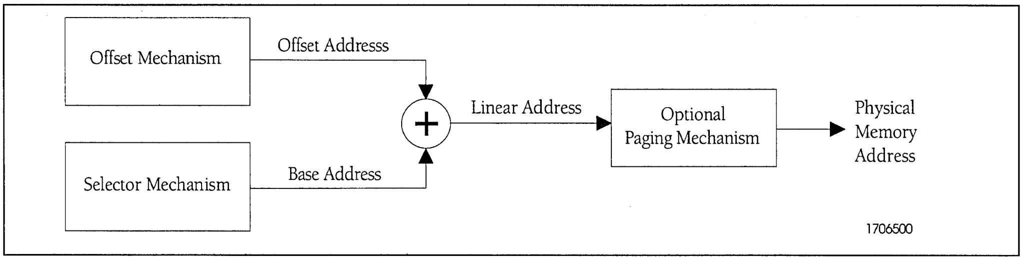


Figure 2-18. Protected Mode Address Calculation

Selector Mechanism

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

The six segment registers (CS, DS, SS, ES, FS and GS) each contain a 16-bit selector that is used when the register is loaded to locate a segment descriptor in either the global descriptor table (GDT) or the local descriptor table (LDT). The segment descriptor defines the base address, limit and attributes of the selected segment and is cached on the Cx486DLC 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 2-19 illustrates the operation of the selector mechanism.

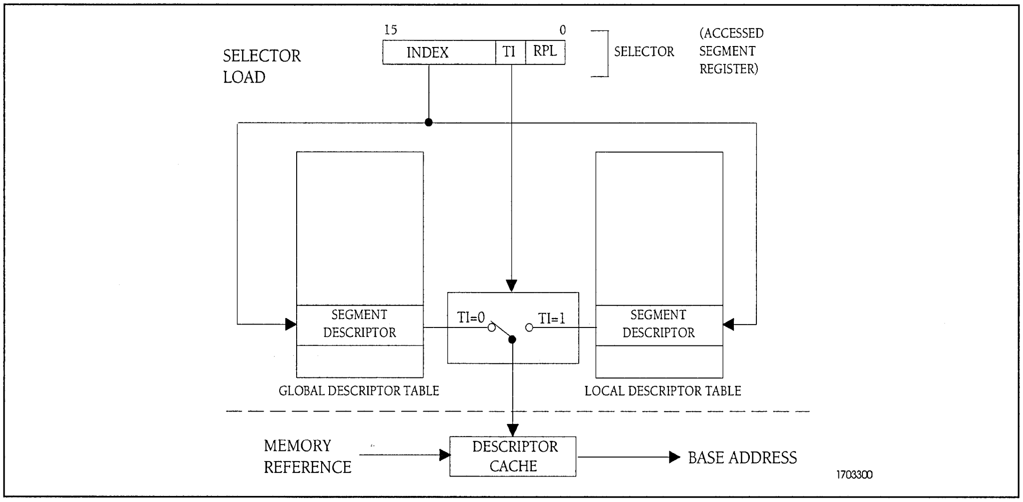


Figure 2-19. Selector Mechanism

Paging Mechanism

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

A page is addressed by using two levels of tables as illustrated in Figure 2-20. The upper 10 bits of the 32-bit linear address are used to locate an entry in the **page directory table**. The page directory table acts as a 32-bit master index to up to 1K individual second-level page tables. The selected entry in the page directory table, referred to as the directory table entry, identifies the starting address of the second-level **page table**. The page directory table itself is a page and is therefore aligned to a 4 KByte boundary. The physical address of the current page directory is stored in the CR3 control register, also referred to as the Page Directory Base Register (PDBR).

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

Since the page directory table can point to 1K page tables, and each page table can point to 1K of page frames, a total of 1M of page frames can be implemented. Since each page frame contains 4 KBytes, up to 4 GBytes of virtual memory can be addressed by the Cx486DLC 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 2-21 and listed in Table 2-14.

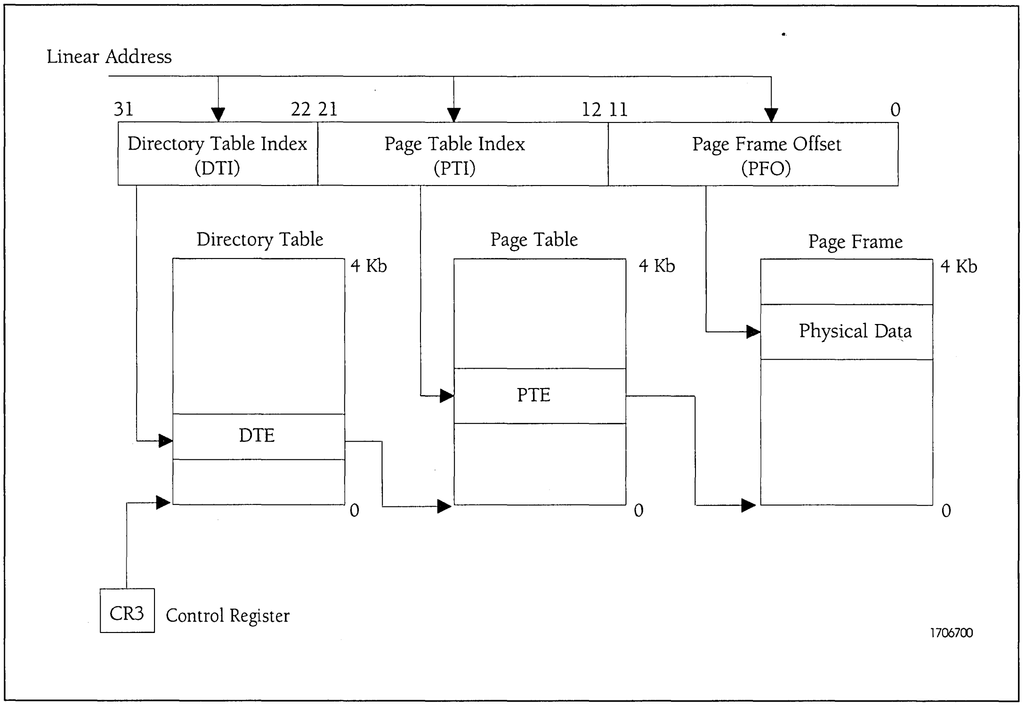


Figure 2-20. Paging Mechanism

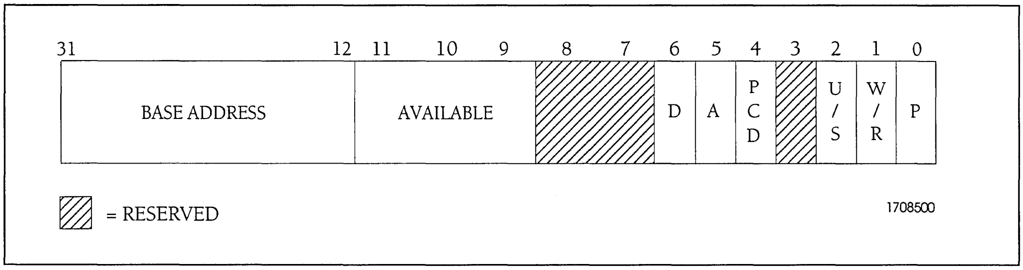


Figure 2-21. Directory and Page Table Entry (DTE and PTE) Format

Table 2-14. 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 the remaining DTE/PTE bits can be used by the programmer.

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

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

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 128 KBytes of memory addresses.

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

2.5 Interrupts and Exceptions

An interrupt or exception changes the sequential flow of a program by transferring program control to a service routine. Both software and hardware interrupts can occur. Software interrupts occur as the result of execution of an INT instruction. Hardware interrupts occur in response to an external interrupt request on the non-maskable interrupt (NMI) or maskable interrupt (INTR) input pins. Exceptions occur as the result of the execution of an instruction that provokes an exception condition. For example, an illegal opcode or a stack fault generates an exception.

When the Cx486DLC services an interrupt or exception, the current program's address and flags are pushed onto the stack to allow resumption of execution of the interrupted program. In protected mode, the processor also saves an error code for some exceptions. Program control is then transferred to the interrupt handler (also called the interrupt service routine). Upon execution of an IRET at the end of the service routine, program execution resumes at the interrupted instruction.

The Cx486DLC accepts up to 256 different interrupts. Each interrupt has a corresponding vector. The vector number is used by the Cx486DLC to locate an entry in the interrupt descriptor table (IDT). In real mode, each IDT entry is a four-byte far pointer to the entry point of the corresponding interrupt service routine. In protected mode, each IDT entry is an eight-byte descriptor. The IDTR register specifies the beginning address and limit of the IDT. Following reset, the IDTR contains a base address of 0h with a limit of 3FFh.

The IDT can be located anywhere in physical memory as determined by the IDTR register. The IDT may contain different types of descriptors: interrupt gates, trap gates and task gates. Interrupt gates are used mostly to enter a hardware interrupt handler. Trap gates are generally used to enter an exception interrupt 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.

Exceptions and the hardware NMI have assigned vectors in the range from 0-31, as shown in Table 2-15. Not all of these first 32 vectors are used by the Cx486DLC, however, unassigned vectors are reserved and should not be used. The vectors for the hardware INTR interrupts are generated by external hardware. In response to an unmasked INTR, the Cx486DLC issues interrupt acknowledge bus cycles used to read the value of the vector from external hardware. Any vector in the range from 32 to 255 may be used. Software INT instructions include the vector as part of the instruction opcode.

2.5.1 Exceptions

Exceptions can be classified in three different categories depending on the way they are reported and if the instruction which first caused the exception can be restarted. Table 2-15 lists the exception type for each of the Cx486DLC exceptions.

Fault exceptions are reported for the current instruction. The instruction is nullified and the fault is reported with the CPU in a state which permits the faulting instruction to be restarted.

Trap exceptions are reported immediately after the execution of the instruction that caused the exception. The instruction pointer restored after execution of the service routine points to the instruction following the instruction that caused the trap. Software interrupt instructions also operate like trap exceptions.

Abort exceptions are caused by a very severe system error. The catastrophic nature of the error does not always allow sufficient information to determine the precise location of the instruction causing the problem and does not allow restart of the program.

Table 2-15. Interrupt Vector Assignments

INTERRUPT VECTOR	FUNCTION	EXCEPTION TYPE
0	Divide error	FAULT
1	Debug exception	TRAP*
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/TRAP
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: Some debug exceptions may report both traps on the previous instruction and faults on the next instruction.

2.5.1.1 Exceptions in Real Mode

Many of the exceptions described in Table 2-15 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 2-16.

Table 2-16. Exception Changes in Real Mode

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

Note: -- = does not occur

2.5.1.2 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 2-22 and the error code bit definitions are listed in Table 2-17. Bits 15-3 (selector index) are not meaningful if the error code was generated as the result of a page fault. The error code is always zero for double faults and alignment check exceptions.

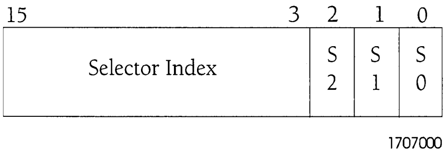


Figure 2-22. Error Code Format

Table 2-17. Error Code Bit Definitions

FAULT TYPE	SELECTOR INDEX (BITS 15-3)	S2 (BIT 2)	S1 (BIT 1)	S0 (BIT 0)
Page Fault	Reserved	Fault caused by: 0 = not present page. 1 = page-level protection vioation.	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 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 exception occurred while trying to invoke exception or hardware interrupt handler.

2.5.2 Hardware Interrupts

Hardware interrupts are classified as either maskable or non-maskable. In most cases, hardware interrupts are serviced after the current instruction is completed. After the interrupt handler is finished, execution continues in the original program with the instruction immediately following the interrupted instruction.

Non-maskable interrupts provide a method of servicing very high priority interrupts. When the NMI input is asserted, the CPU automatically transfers program control to the interrupt service routine corresponding to vector 2. Since the interrupt vector is fixed and is supplied internally, no interrupt acknowledge bus cycles are performed.

While executing the NMI service routine, the Cx486DLC microprocessor does not service any further NMI requests until an interrupt return (IRET) instruction is executed or the processor is reset. If another NMI occurs while currently servicing an NMI, its presence is saved for servicing after execution of the next IRET instruction. It is recommended that an interrupt gate be used for the NMI in order to disable nested maskable interrupts. Execution of an IRET instruction in the maskable interrupt handler allows the NMI to be re-enabled.

Hardware maskable interrupts occur when the INTR pin is asserted and the Interrupt Enable Flag (IF) bit is set to 1 in the EFLAGS register. The processor only responds to maskable interrupts between instructions (string instructions have an interrupt window between memory moves that allows interrupts during long string moves). When an interrupt occurs, the processor reads an 8-bit vector supplied by external system hardware. This vector selects which of the 256 possible interrupt handlers is executed in response to the interrupt.

2.5.3 Software Interrupts

The third type of interrupt/exception for the Cx486DLC microprocessor is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table. Execution of the interrupt service routine occurs regardless of the state of the IF flag in the EFLAGS register.

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

The last type of software interrupt is the single-step trap. The single-step feature is enabled by setting the TF flag in the EFLAGS register. This causes the processor to generate a debug exception after the execution of every instruction.

2.5.4 Interrupt and Exception Priorities

Hardware interrupts are generated external to the CPU. Maskable interrupts produced on the INTR pin and non-maskable interrupts produced on the NMI input are recognized between instructions. When NMI and maskable INTR interrupts are both detected at the same instruction boundary, the Cx486DLC microprocessor services the NMI interrupt first.

Exceptions are generated internal to the CPU. The Cx486DLC checks for exceptions in parallel with instruction decoding and execution. Several exceptions can result from a single instruction. However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should make the appropriate corrections to the instruction and then restart the instruction. In this way, exceptions can be serviced until the instruction executes properly.

The Cx486DLC supports instruction restart after all faults, except when an instruction causes a task switch to a task whose task state segment (TSS) is partially not present. A TSS can be partially not present if the TSS is not page aligned and one of the pages where the TSS resides is not currently in memory.

As the Cx486DLC executes instructions, it follows a consistent policy for prioritizing exceptions and hardware interrupts as listed in Table 2-18.

Table 2-18. 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	Non-maskable hardware interrupt.	Caused by NMI asserted.
4	Maskable hardware interrupt.	Caused by INTR asserted and IF = 1.
5	Faults resulting from fetching the next instruction.	Includes segment not present, general protection fault and page fault.
6	Faults resulting from instruction decoding.	Includes illegal opcode, instruction too long, or privilege violation.
7	WAIT instruction and TS = 1 and MP = 1.	Device not available exception generated.
8	ESC instruction and EM = 1 or TS = 1.	Device not available exception generated.
9	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.	

2.6 Shutdown and Halt

The **halt instruction** (HLT) stops program execution and prevents the processor from using the local bus until restarted. The Cx486DLC then enters a low-power suspend mode. NMI, INTR with interrupts enabled (IF bit in EFLAGS=1), or RESET forces the CPU out of the halt state. If interrupted, the saved code segment and instruction pointer specify the instruction following the HLT.

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

2.7 Protection

Segment protection and page protection are safeguards built into the Cx486DLC protected mode architecture which deny unauthorized or incorrect access to selected memory addresses. These safeguards allow multitasking programs to be isolated from each other and from the operating system. Page protection is discussed earlier in this chapter in section 2.4. This section concentrates on segment protection.

Selectors and descriptors are the key elements in the segment protection mechanism. The segment base address, size, and privilege level are established by a segment descriptor. Privilege levels control the use of privileged instructions, I/O instructions and access to segments and segment descriptors. Selectors are used to locate segment descriptors. Segment accesses are divided into two basic types, those involving code segments (e.g, control transfers) and those involving data accesses. The ability of a task to access a segment depends on:

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

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

2.7.1 Privilege Levels

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

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

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

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

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

2.7.2 I/O Privilege Levels

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

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

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

2.7.3 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 sevicing and from execution of the CALL, JMP, INT, IRET and RET instructions.

There are five types of control transfers that are summarized in Table 2-19. 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 2-19. Descriptor Types Used for Control Transfer

TYPE OF CONTROL TRANSFER	OPERATION TYPES	DESCRIPTOR REFERENCED	DESCRIPTOR TABLE
Intersegment within the same privilege level.	JMP, CALL, RET, IRET*	Code Segment	GDT or LDT
Intersegment to the same or a more privileged level. Interrupt within task (could change CPL level).	CALL	Call Gate	GDT or LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
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.

2.7.3.1 Gates

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

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

2.7.4 Initialization and Transition To Protected Mode

The Cx486DLC microprocessor switches to Real Mode immediately after RESET. While operating in real mode, the system tables and registers should be initialized. The GDTR and IDTR must point to a valid GDT and IDT, respectively. The size of the IDT should be at least 256 bytes, and the GDT must contain descriptors which describe the initial code and data segments.

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

2.8 Virtual 8086 Mode

Both Real Mode and Virtual 8086 (V86) Mode are supported by the Cx486DLC CPU allowing execution of 8086 application programs and 8086 operating systems. V86 Mode allows the execution of 8086-type applications, yet still permits use of the Cx486DLC protection mechanism. V86 tasks run at privilege level 3. Upon entry, all segment limits are set to FFFFh (64K) as in real mode.

2.8.1 Memory Addressing

While in V86 mode, segment registers are used in an identical fashion to Real Mode. The contents of the segment register are multiplied by 16 and added to the offset to form the segment base linear address.

The Cx486DLC CPU permits the operating system to select which programs use the V86 address mechanism and which programs use protected mode addressing for each task.

The Cx486DLC also permits the use of paging when operating in V86 mode. Using paging, the 1-MByte address space of the V86 task can be mapped to anywhere in the 4-GByte linear address space of the Cx486DLC CPU. As in real mode, linear addresses that exceed 1 MByte cause a segment limit overrun exception.

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

2.8.2 Protection

All V86 tasks operate with the least amount of privilege (level 3) and are subject to all of the Cx486DLC protected mode protection checks. As a result, any attempt to execute a privileged instruction within a V86 task results in a general protection fault.

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

2.8.3 Interrupt Handling

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

2.8.4 Entering and Leaving V86 Mode

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

3. BUS INTERFACE

3.1 Overview

The following paragraphs describe the Cx486DLC input and output signals. The discussion of these signals is arranged by functional groups as shown in Figure 3-1. Table 3-1 gives a brief description of each of the Cx486DLC signals.

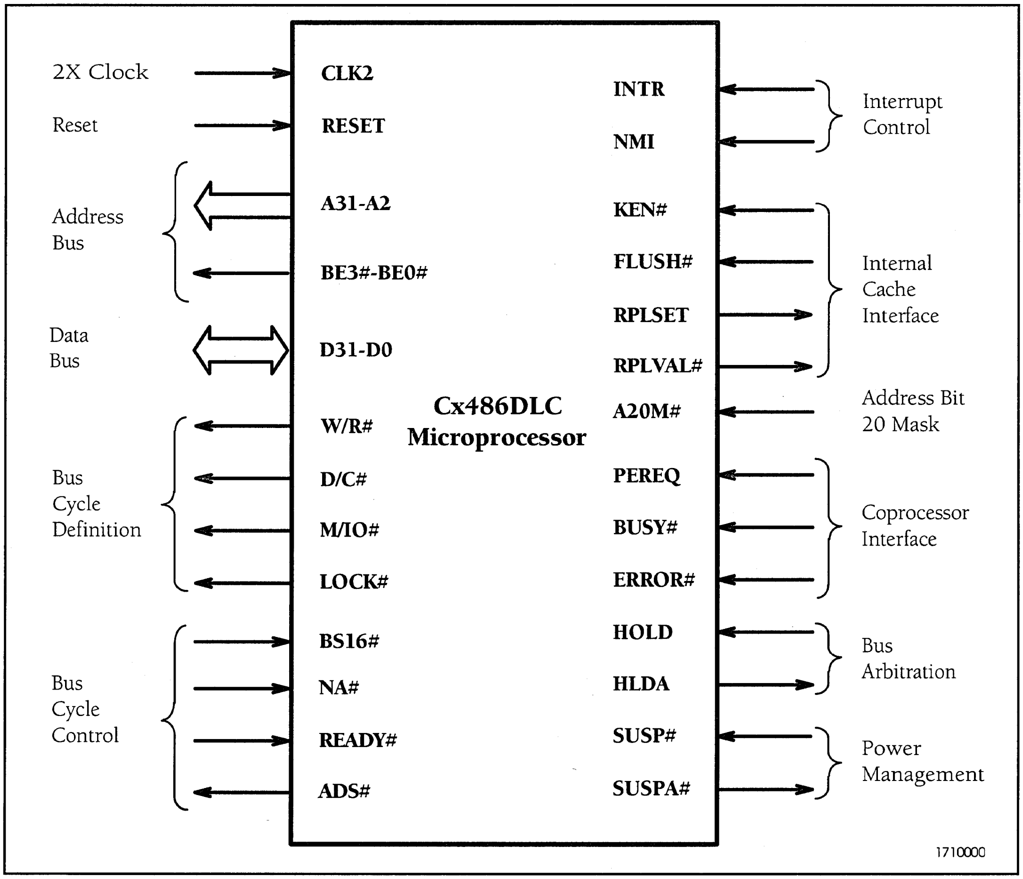


Figure 3-1. Cx486DLC Functional Signal Groupings

Table 3-1. Cx486DLC Signal Summary

SIGNAL	SIGNAL NAME	SIGNAL GROUP
A20M#	ADDRESS BIT 20 MASK	
A31-A2	ADDRESS BUS LINES	Address Bus
ADS#	ADDRESS STROBE	Bus Cycle Control
BE3# - BE0#	BYTE ENABLES	Address Bus
BS16#	BUS SIZE 16	Bus Cycle Control
BUSY#	PROCESSOR EXTENSION BUSY	Coprocessor Interface
CLK2	2X CLOCK INPUT	
D31-D0	DATA BUS	
D/C#	DATA/CONTROL	Bus Cycle Definition
ERROR#	PROCESSOR EXTENSION ERROR	Coprocessor Interface
FLUSH#	CACHE FLUSH	Internal Cache Interface
INTR	MASKABLE INTERRUPT REQUEST	Interrupt Control
HLDA	HOLD ACKNOWLEDGE	Bus Arbitration
HOLD	HOLD REQUEST	Bus Arbitration
LOCK#	BUS LOCK	Bus Cycle Definition
KEN#	CACHE ENABLE	Internal Cache Interface
M/IO#	MEMORY/INPUT-OUTPUT	Bus Cycle Definition
NA#	NEXT ADDRESS REQUEST	Bus Cycle Control
NMI	NON-MASKABLE INTERRUPT REQUEST	Interrupt Control
PEREQ	PROCESSOR EXTENSION REQUEST	Coprocessor Interface
READY#	BUS READY	Bus Cycle Control
RESET	RESET	
RPLSET	REPLACEMENT SET	Internal Cache Interface
RPLVAL#	REPLACEMENT SET VALID	Internal Cache Interface
SUSP#	SUSPEND REQUEST	Power Management
SUSPA#	SUSPEND ACKNOWLEDGE	Power Management
W/R#	WRITE/READ	Bus Cycle Definition

The "#" (pound) symbol following a signal name indicates that when the signal is in its active (asserted) state, the signal is at a logic low level. When the # is not present at the end of a signal name, the logic high level represents the active state. The following two sections describe the signals and their functional timing characteristics.

Additional signal information may be found in Chapter 4, Electrical Specifications. Chapter 4 documents the DC and AC characteristics for the signals including voltage levels, propagation delays, setup times and hold times. Specified setup and hold times must be met for proper operation of the Cx486DLC.

3.2 Signal Descriptions

3.2.1 2X Clock Input

The **2X Clock Input (CLK2)** signal is the basic timing reference for the Cx486DLC microprocessor. 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.

3.2.2 Reset

Reset is an active high input signal that, when asserted, suspends all operations in progress and places the Cx486DLC into a reset state. RESET is a level-sensitive synchronous input and must meet specified setup and hold times to be recognized by the Cx486DLC properly. The Cx486DLC begins executing instructions at physical address location FFFF FFF0h approximately 400 CLK2s after RESET is driven inactive (low). While RESET is active 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 Cx486DLC are shown in Table 3-2.

Table 3-2. Signal States During Reset

SIGNAL NAME	SIGNAL STATE DURING RESET
A20M#	Ignored
A31-A2	1
ADS#	1
BE3# - BE0#	0
BS16#	Ignored
BUSY#	Ignored
D31-D0	Float
D/C#	1
ERROR#	Ignored
FLUSH#	Ignored
HLDA	0
HOLD	Ignored
INTR	Ignored
KEN#	Ignored
LOCK#	1
M/IO#	0
NA#	Ignored
NMI	Ignored
PEREQ	Ignored
READY#	Ignored
RESET	Input Recognized
RPLSET	Float
RPLVAL#	Float
SUSP#	Ignored
SUSPA#	Float
W/R#	0

3.2.3 Address Bus

The **Address Bus (A31-A2)** signals are three-state outputs that provide physical memory or I/O port addresses. All address lines can be used for addressing physical memory allowing a 4-GByte address space (0000 0000h to FFFF FFFFh). During I/O port accesses, except for coprocessor accesses, A31-A16 are driven low. This allows for a 64 KByte I/O address space (0000 0000h to 0000 FFFFh).

During coprocessor I/O accesses, A30 - A16 are driven low and A31 is driven high to allow it to be used by external logic to generate a coprocessor select signal. Consequently, for coprocessor I/O cycles the Cx486DLC drives address 8000 00F8h with command transfers and address 8000 00FCh with data transfers.

Address lines A31 - A2 float while the CPU is in hold acknowledge.

Byte Enables (BE3# - BE0#), shown in Table 3-3, are three-state outputs that determine which bytes within the 32-bit data bus will be transferred during a memory or I/O access. During a memory write, one or both of the upper bytes (D and C) of the data bus may be duplicated in the lower bytes (B and A) of the bus. This duplication is dependent on BE3#-BE0# as listed in Table 3-4.

Table 3-3. Byte Enable Line Definitions

BYTE ENABLE LINE	BYTE TRANSFERRED
BE0#	D7 - D0
BE1#	D15 - D8
BE2#	D23 - D16
BE3#	D31 - D24

Table 3-4. 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 Cx486DLC 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 = not defined

Generating A1-A0 using BE3#-BE0# can be achieved by using the following equations:

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

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

The relationship between A1-A0 and BE3#-BE0# is shown in Table 3-5.

Table 3-5. 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

x = don't care

3.2.4 Data Bus

The **Data Bus (D31 - D0)** signals are three-state bidirectional signals which provide the data path between the Cx486DLC and external memory and I/O devices. The data bus inputs data during memory read, I/O read and interrupt acknowledge cycles and outputs data during memory and I/O write cycles. Data read operations require that specified data setup and hold times be met for correct operation. The data bus signals are high active and float while the CPU is in a hold acknowledge state.

3.2.5 Bus Cycle Definition

The bus cycle definition signals consist of four three-state outputs (M/IO#, D/C#, W/R#, LOCK#). These outputs define the bus cycle type for the operation being performed as listed in Table 3-6. M/IO#, D/C# and W/R# are the primary bus cycle definition signals and are valid when ADS# (Address Strobe) is active. During non-pipelined cycles, the LOCK# output is driven valid along with M/IO#, D/C# and W/R#. During pipelined addressing, LOCK# is driven valid 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 Cx486DLC is in a hold acknowledge state.

Memory/IO (M/IO#) distinguishes between memory and I/O operations. This signal indicates that the current bus cycle is a memory read or write. When M/IO# is not asserted the current bus cycle is an I/O read or write.

Data/Control (D/C#) distinguishes between data and control operations. This signal indicates that the current bus cycle is a data transfer to or from memory or I/O. When D/C# is not asserted the current bus cycle involves a control function such as halt, interrupt servicing or code fetch.

Write/Read (W/R#) distinguishes between write and read bus cycles. This signal indicates that the current bus cycle is a write to memory or I/O. When W/R# is not asserted the current bus cycle is a memory or I/O read operation.

LOCK# is an active low output which, when asserted, indicates that other system bus masters are denied access to control of the system bus. The LOCK# signal may be explicitly activated during bus operations by including the LOCK prefix on certain instructions. LOCK# is always asserted during descriptor updates, interrupt acknowledge sequences and when executing the XCHG instruction. The Cx486DLC does not enter the hold acknowledge state in response to HOLD while the LOCK# input is active.

Table 3-6. 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, BE3# - BE0# = 1011 Shutdown: A31 - A2 = 0h, 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

3.2.6 Bus Cycle Control

The bus cycle control signals (ADS#, BS16#, NA#, READY#) allow the Cx486DLC to indicate the beginning of a bus cycle and allow system hardware to control address pipelining, bus cycle termination timing, and bus sizing.

Address Strobe (ADS#) is an active low, three-state output which indicates that the Cx486DLC has driven a valid address (A31 - A2, BE3# - BE0#) and bus cycle definition (M/IO#, D/C#,W/R#) on the appropriate Cx486DLC output pins. During non-pipelined 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 Cx486DLC is in a hold acknowledge state.

Ready (READY#) is an active low input which is driven 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 Cx486DLC latches the input data and terminates the cycle. During a write cycle, READY# assertion indicates that the system hardware has accepted the Cx486DLC output data. READY# must be asserted to terminate every bus cycle, including halt and shutdown indication cycles.

Next Address Request (NA#) is an active low input used to request address pipelining. Assertion of this input by the system hardware indicates it is prepared to accept new bus cycle definition and address signals (M/IO#, D/C#, W/R#, A31-A2, BE3# - BE0#) from the microprocessor even if the current bus cycle has not been terminated by assertion of READY#. If the Cx486DLC has an internal bus request pending and the NA# input is sampled active, the next bus definition and address signals are driven onto the bus.

Bus Size 16 (BS16#) is an active low input that allows connection of the 32-bit Cx486DLC 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.

3.2.7 Interrupt Control

The interrupt control input signals (INTR, NMI) allow the execution of the Cx486DLC's current instruction stream to be interrupted and suspended.

Maskable Interrupt Request (INTR) is an active high level-sensitive input which causes the processor to suspend execution of the current instruction stream and begin execution of an interrupt service routine. The INTR input can be masked (ignored) through the Flags Register IF bit. When not masked, the Cx486DLC responds to the INTR input by performing two locked interrupt acknowledge bus cycles. The second interrupt acknowledge cycle reads an 8-bit value, the interrupt vector, from an external interrupt controller. The 8-bit interrupt vector indicates the interrupt level that caused generation of the INTR and is used by the CPU to determine the beginning address of the interrupt service routine. To assure recognition of the INTR request, INTR must remain active until the start of the first interrupt acknowledge cycle.

Non-maskable Interrupt (NMI) Request is a rising edge sensitive input which causes the processor to suspend execution of the current instruction stream and begin execution of an NMI interrupt service routine. The NMI interrupt service request cannot be masked by software. Asserting NMI causes an interrupt which 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 after execution of the next IRET instruction (typically at the end of the NMI service routine). If NMI is re-asserted prior to execution of the IRET instruction, one and only one NMI rising edge is stored and is processed after execution of the next IRET. 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 Cx486DLC restarts execution only in response to RESET or an unmasked INTR. NMI does not restart CPU execution under this condition.

The Cx486DLC samples NMI at the beginning of each phase 2. 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 guarantee recognition at a particular clock edge.

3.2.8 Internal Cache Interface

The internal cache interface signals (KEN#, FLUSH#, RPLSET, RPLVAL#) are used to indicate cache status and control caching activity.

Cache Enable (KEN#) is an active low input which indicates that the data being returned during the current cycle is cacheable. When KEN# is active and the Cx486DLC 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 non-cacheable region registers can also be used to control cacheability. Memory addresses specified by the non-cacheable region registers are not cacheable regardless of the state of KEN#. I/O accesses, locked reads and interrupt acknowledge cycles are never cached.

During any 32-bit cache fill cycle with KEN# asserted, the Cx486DLC ignores the state of the byte enables (BE3# - BE0#) and always writes four bytes of data into the cache. If BS16# is asserted during a cache fill cycle that requires two 16-bit transfers and KEN# is active, KEN# must be asserted for both 16-bit cycles to cause a cache fill to occur. The KEN# input is ignored following reset and can be enabled using the KEN bit in the CCR0 configuration register.

Cache Flush (FLUSH#) is an active low input which 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.

Replacement Set (RPLSET) is an output indicating which set in the cache is currently undergoing a line replacement. This signal is meaningful only when the internal cache is configured as two-way set associative. The RPLSET output is disabled (three-state) following reset and can be enabled using the RPL bit in the CCR1 configuration register.

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

3.2.9 Address Bit 20 Mask

Address Bit 20 Mask (A20M#) is an active low input which causes the Cx486DLC to mask (force low) physical address bit 20 when driving the external address bus or performing an internal cache access. Asserting A20M# emulates the 1 MByte address wrap around 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.

3.2.10 Coprocessor Interface

The data bus, address bus and bus cycle definition signals, as well as the coprocessor interface signals (PEREQ, BUSY#, ERROR#), are used to control communication between the Cx486DLC and a coprocessor. Coprocessor or ESC opcodes are decoded by the Cx486DLC and the opcode and operands are then transferred to the coprocessor via I/O port accesses to addresses 8000 00F8h, and 8000 00FCh. 8000 00F8h functions as the control port address and 8000 00FCh is used for operand transfers. Additional handshaking is provided using the three dedicated control signals described below.

Coprocessor Request (PEREQ) is an active high input which indicates the coprocessor is ready to transfer data to or from the CPU. The coprocessor may assert PEREQ in the process of executing a coprocessor instruction. The Cx486DLC internally stores the current coprocessor opcode and performs the correct data transfers to support coprocessor operations using PEREQ to synchronize the transfer of required operands. PEREQ is internally connected to a pull-down resistor to prevent this signal from floating active when left unconnected.

Coprocessor Busy (BUSY#) is an active low input from the coprocessor which indicates to the Cx486DLC that the coprocessor is currently executing an instruction and is not yet able to accept another opcode. When the Cx486DLC processor encounters a WAIT instruction or any coprocessor instruction which operates on the coprocessor stack (e.g. load, pop, arithmetic operation), BUSY# is sampled. BUSY# is continually sampled and must be recognized as inactive before the CPU will supply the coprocessor with another instruction. However, the following coprocessor instructions are allowed to execute even if BUSY# is active since these instructions are used for coprocessor initialization and exception clearing:

FNINIT
FNCLEX

BUSY# is internally connected to a pull-up resistor to prevent it from floating active when left unconnected.

Coprocessor Error (ERROR#) is an active low input used to indicate that the coprocessor generated an error during execution of a coprocessor instruction. ERROR# is sampled by the Cx486DLC processor 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.

Certain coprocessor instructions do not generate an exception 16 even if ERROR# is active. These instructions, which involve clearing coprocessor error flags and saving the coprocessor state, are listed below:

FNINIT
FNCLEX
FNSTSW
FNSTCW
FNSTENV
FNSAVE

ERROR# is internally connected to a pull-up resistor to prevent it from floating active when left unconnected.

3.2.11 Bus Arbitration

The bus arbitration (HOLD, HLDA) signals allow the Cx486DLC to relinquish control of its local bus when requested by another bus master device. Once the processor has relinquished its bus (three-stated), the bus master device can then drive the local bus signals.

Hold Request (HOLD) is an active high input used to indicate that another bus master requests control of the local bus. After recognizing the HOLD request and completing the current bus cycle or sequence of locked bus cycles, the Cx486DLC responds by floating (three-state) the local bus and asserting the hold acknowledge (HLDA) output. A31-A2, ADS#, BE3#-BE0#, D31-D0, D/C#, LOCK#, M/IO#, RPLSET, RPLVAL# and W/R# are floated while HLDA is asserted.

Once HLDA is asserted, the bus remains granted to the requesting bus master until HOLD becomes inactive. When the Cx486DLC recognizes HOLD is inactive, it simultaneously drives the local bus and drives HLDA inactive. External pull-up resistors may be required on some of the Cx486DLC tri-state outputs to guarantee 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 Cx486DLC places the bus into an idle state instead of a hold acknowledge state. The HOLD input is also recognized during suspend mode provided the CLK2 input has not been stopped. HOLD is level-sensitive and must meet specified setup and hold times for correct operation.

Hold Acknowledge (HLDA) is an active high output which indicates that the Cx486DLC is in a hold acknowledge state and has relinquished control of its local bus. While in the hold acknowledge state, the Cx486DLC drives HLDA active and continues to drive SUSPA#. The other Cx486DLC outputs, A31-A2, ADS#, BE3#-BE0#, D31-D0, D/C#, LOCK#, M/IO#, RPLSET, RPLVAL# and W/R# are in a high-impedance state allowing the requesting bus master to drive these signals. If the on-chip cache can satisfy bus requests, the Cx486DLC continues to operate during hold acknowledge states.

The processor deactivates HLDA when the HOLD request is driven inactive. The Cx486DLC stores one 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 Cx486DLC may or may not enter suspend mode depending on the state of the internal execution pipeline.

Table 3-7 summarizes the state of the Cx486DLC output signals during hold acknowledge.

Table 3-7. Signal States During Hold Acknowledge

SIGNAL NAME	SIGNAL STATE DURING HOLD ACKNOWLEDGE
A20M#	Recongnized Internally
A31-A2	Float
ADS#	Float
BE3# - BE0#	Float
BS16#	Ignored
BUSY#	Ignored
D31-D0	Float
D/C#	Float
ERROR#	Ignored
FLUSH#	Input Recognized
HLDA	1
HOLD	Input Recognized
INTR	Ignored
KEN#	Ignored
LOCK#	Float
M/IO#	Float
NA#	Ignored
NMI	Input Recognized
PEREQ	Ignored
READY#	Ignored
RESET	Input Recognized
RPLSET	Float
RPLVAL#	Float
SUSP#	Input Recognized
SUSPA#	Driven
W/R#	Float

3.2.12 Power Management

Two power management signals allow the Cx486DLC to enter and exit suspend mode. Suspend mode can also be entered as the result of executing a HALT instruction. Suspend mode circuitry allows the Cx486DLC to consume minimal power while maintaining the entire internal CPU state.

Suspend Request (SUSP#) is an active low input which requests that the Cx486DLC enter suspend mode. After recognizing SUSP# is active, the processor completes execution of the current instruction, any pending decoded instructions and associated bus cycles. In addition, the Cx486DLC 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 associated with monitoring RESET, HOLD and FLUSH# remains active. With SUSPA# asserted, the CLK2 input to the Cx486DLC can be stopped in either phase. Stopping the CLK2 input further reduces current consumption of the Cx486DLC.

To resume operation, the CLK2 input is restarted (if stopped), followed by deassertion of the SUSP# input. The processor then resumes instruction fetching and begins execution in the instruction stream at the point it had stopped. The SUSP# input is level-sensitive and must meet specified setup and hold times to be recognized at a particular clock edge. The SUSP# input is ignored following reset and can be enabled using the SUSP bit in the CCR0 configuration register.

The **Suspend Acknowledge (SUSPA#)** output indicates that the Cx486DLC has entered 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 Cx486DLC continues to recognize RESET, HOLD and FLUSH#. If suspend mode was entered as the result of a HALT instruction, the

Cx486DLC also continues to monitor the NMI input and an unmasked INTR input. Detection of INTR or NMI forces the Cx486DLC to exit suspend mode and begin execution of the appropriate interrupt service routine. The CLK2 input to the processor may be stopped after SUSPA# has been asserted to further reduce the current consumption of the Cx486DLC. The SUSPA# output is disabled (floated) following reset and can be enabled using the SUSP bit in the CCR0 configuration register.

Table 3-8 shows the state of the Cx486DLC signals when the device is in suspend mode.

Table 3-8. Signal States During Suspend Mode

SIGNAL NAME	SIGNAL STATE DURING SUSP# INITIATED SUSPEND MODE	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
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	Ignored	Input Recognized
PEREQ	Ignored	Ignored
READY#	Ignored	Ignored
RESET	Input Recognized	Input Recognized
RPLSET	Driven	Driven
RPLVAL#	Driven	Driven
SUSP#	Input Recognized	Ignored
SUSPA#	1	1
W/R#	0	0

3.3 Functional Timing

3.3.1 Reset Timing and Internal Clock Synchronization

RESET is the highest priority input signal and is capable of interrupting any processor activity when it is asserted. When RESET is asserted, the Cx486DLC aborts any bus cycle. Idle, hold acknowledge and suspend states are also discontinued and the reset state is established. RESET is used when the Cx486DLC microprocessor is powered up to initialize the CPU to a known valid state and to synchronize the internal CPU clock with external clocks.

RESET must be asserted for at least 15 CLK2 periods to ensure recognition by the Cx486DLC microprocessor. If the self-test feature is to be invoked, RESET must be asserted for at least 80 CLK2 periods. RESET pulses less than 15 CLK2 periods may not have sufficient time to propagate throughout the Cx486DLC and may not be recognized. RESET pulses less than 80 CLK2 periods followed by a self-test request may incorrectly report a self-test failure when no true failure exists.

Provided the RESET falling edge meets specified setup and hold times, the internal processor clock phase is synchronized as illustrated in Figure 3-2. The internal processor clock is half the frequency of the CLK2 input and each CLK2 cycle corresponds to an internal CPU clock phase. Phase 2 of the internal clock is defined to be the second rising edge of CLK2 following the falling edge of RESET.

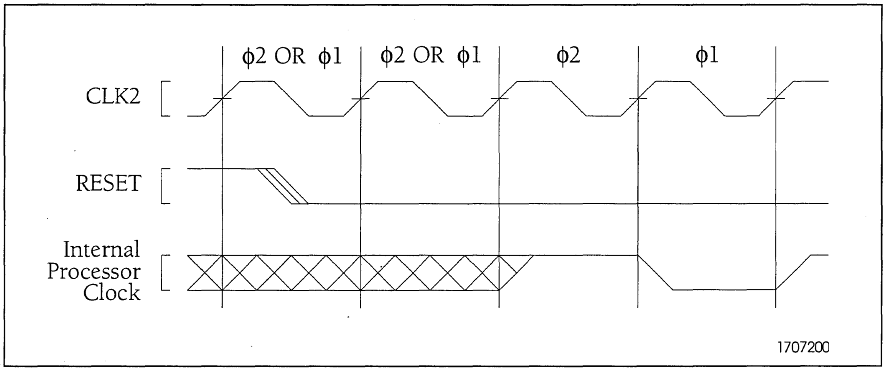


Figure 3-2. Internal Processor Clock Synchronization

Following the falling edge of RESET (and after self-test if it was requested), the Cx486DLC CPU performs an internal initialization sequence for approximately 400 CLK2 periods. The Cx486DLC self-test feature is invoked if the BUSY# input is an 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 Cx486DLC microprocessor attempts to proceed with the reset sequence. Figure 3-3 illustrates the bus activity and timing during the Cx486DLC reset sequence.

Upon completion of self-test, the EAX register contains 0000 0000h if the Cx486DLC microprocessor passed its internal self-test with no problems detected. Any non-zero value in the EAX register indicates that the microprocessor is faulty.

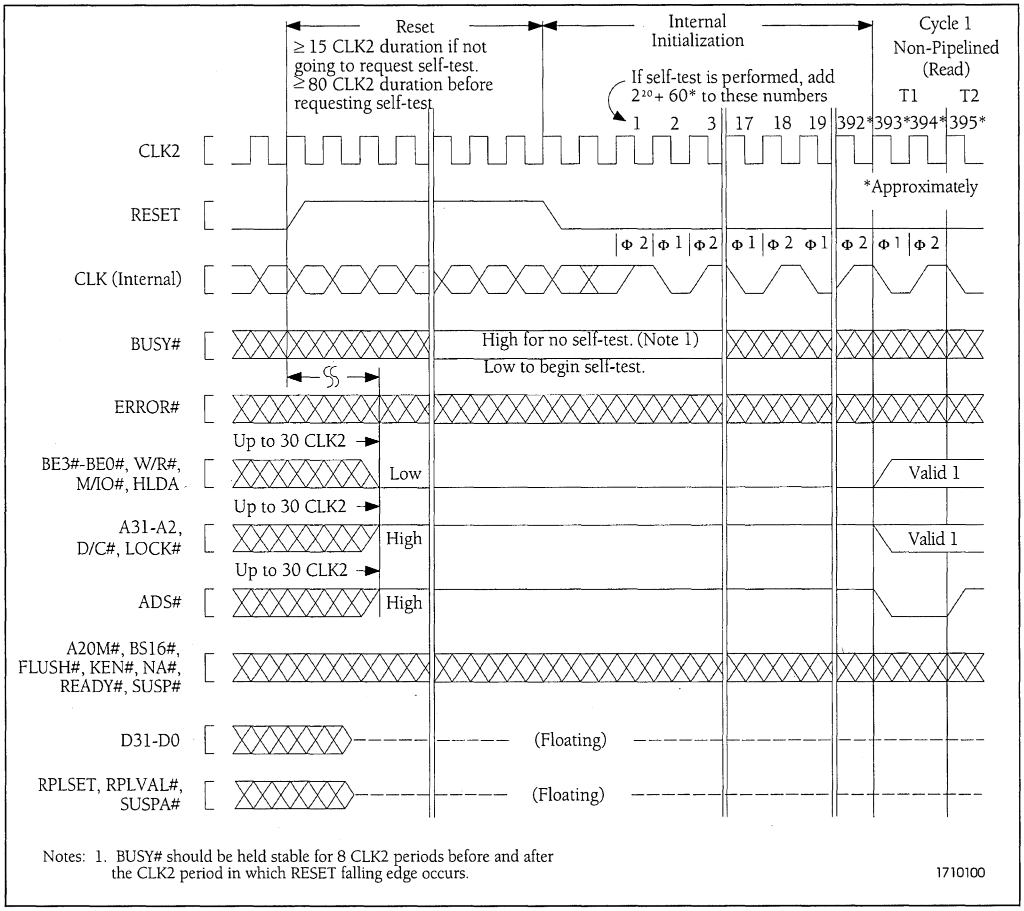

Reset
≥ 15 CLK2 duration if not going to request self-test.
≥ 80 CLK2 duration before requesting self-test
Internal Initialization
If self-test is performed, add 2^{20}+ 60* to these numbers
Cycle 1 Non-Pipelined (Read)
T1
T2
1 2 3 17 18 19 392* 393* 394* 395*
*Approximately
CLK2
RESET
CLK (Internal)
Φ 2 Φ 1 Φ 2 Φ 1 Φ 2 Φ 1 Φ 2 Φ 1 Φ 2
BUSY#
High for no self-test. (Note 1)
Low to begin self-test.
ERROR#
Up to 30 CLK2
BE3#-BE0#, W/R#, M/IO#, HLDA
Low
Valid 1
A31-A2, D/C#, LOCK#
High
ADS#
A20M#, BS16#, FLUSH#, KEN#, NA#, READY#, SUSP#
D31-D0
(Floating)
RPLSET, RPLVAL#, SUSPA#
Notes: 1. BUSY# should be held stable for 8 CLK2 periods before and after the CLK2 period in which RESET falling edge occurs.
1710100

Figure 3-3. Bus Activity from RESET until First Code Fetch

3.3.2 Bus Operation

The Cx486DLC 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.

Cx486DLC instructions can act on memory data operands consisting of 8-bit bytes, 16-bit words or 32-bit double words. The Cx486DLC bus architecture allows for bus transfers of these operands without restrictions on physical address alignment. Any byte boundary alignment is permissible. Operands not aligned on a word boundary may require more than one bus cycle to transfer the operand. This feature is transparent to the programmer.

The Cx486DLC data bus (D31-D0) is a bidirectional bus which 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 space are physically addressed as arrays of 32-bit double words.

The Cx486DLC drives the data bus during write bus cycles and the external system hardware drives the data bus during read bus cycles.
Every bus cycle begins with the assertion of the address strobe (ADS#). ADS# indicates that the Cx486DLC 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 if a memory or I/O operation is occurring, W/R# defines the cycle to be 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 Cx486DLC 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 Cx486DLC 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.

3.3.2.1 Bus Cycles using Non-Pipelined Addressing

Non-Pipelined Bus States

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

The first state of a non-pipelined 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 non-pipelined cycle is called T2. T2 terminates a bus cycle with the assertion of the READY# input and valid data is either input or output depending on the bus cycle type. The fastest Cx486DLC 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.

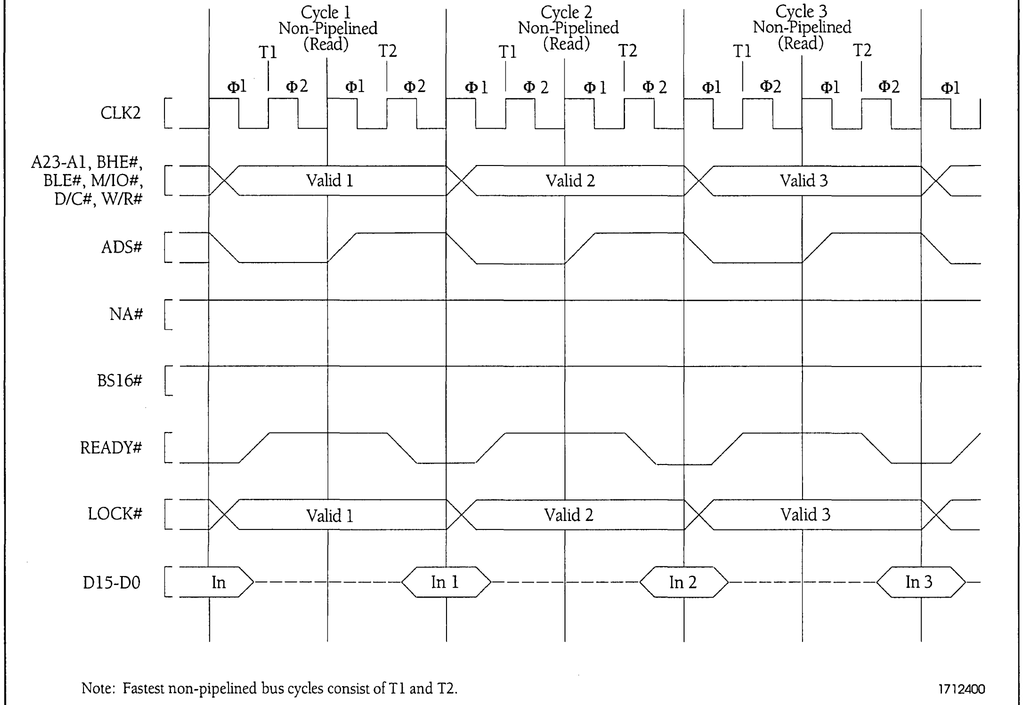
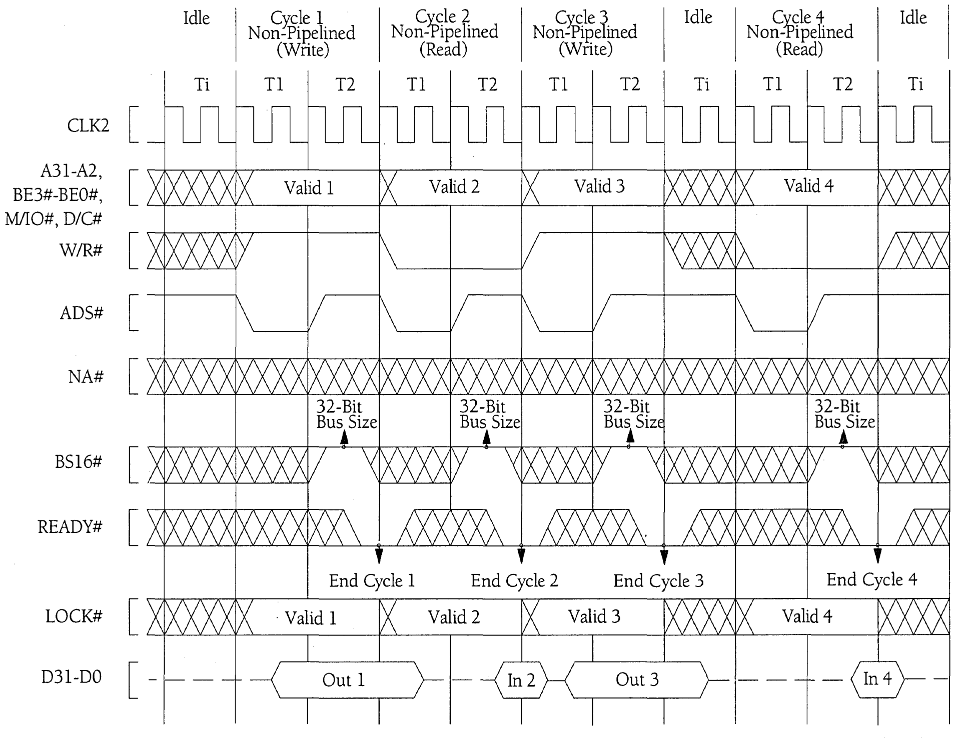

Cycle 1 Non-Pipelined (Read)
T1
T2
Cycle 2 Non-Pipelined (Read)
Cycle 3 Non-Pipelined (Read)
Φ1
Φ2
CLK2
A23-A1, BHE#, BLE#, M/IO#, D/C#, W/R#
Valid 1
Valid 2
Valid 3
ADS#
NA#
BS16#
READY#
LOCK#
D15-D0
In
In 1
In 2
In 3
Note: Fastest non-pipelined bus cycles consist of T1 and T2.
1712400

Figure 3-4. Fastest Non-Pipelined Read Cycles

Non-Pipelined Read and Write Cycles

Any bus cycle may be performed with non-pipelined address timing. Figure 3-5 shows a mixture of read and write cycles with non-pipelined address timing. When a read cycle is performed, the Cx486DLC microprocessor floats its data bus and the externally addressed device then drives the data. The Cx486DLC 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 Cx486DLC CPU 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 Cx486DLC CPU beginning in phase two of T1. When a write cycle is acknowledged, the Cx486DLC write data remains valid throughout phase one of the next bus state to provide write data hold time.



Note: Idle states are shown here for diagram variety only.

1710300

Figure 3-5. Various Non-Pipelined Bus Cycles (no wait states)

Non-Pipelined Wait States

Once a bus cycle begins, it continues until acknowledged by the external system hardware using the Cx486DLC 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. If READY# is not immediately asserted however, 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, it withholds the READY# signal and at least one wait state is added 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 non-pipelined 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 Cx486DLC ignores the READY# input at the end of the T1 state.

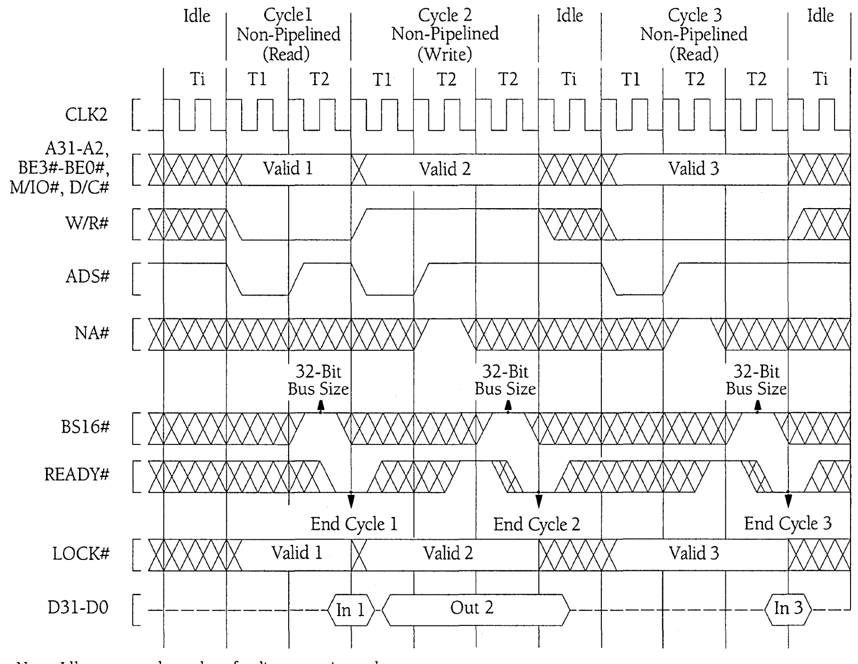
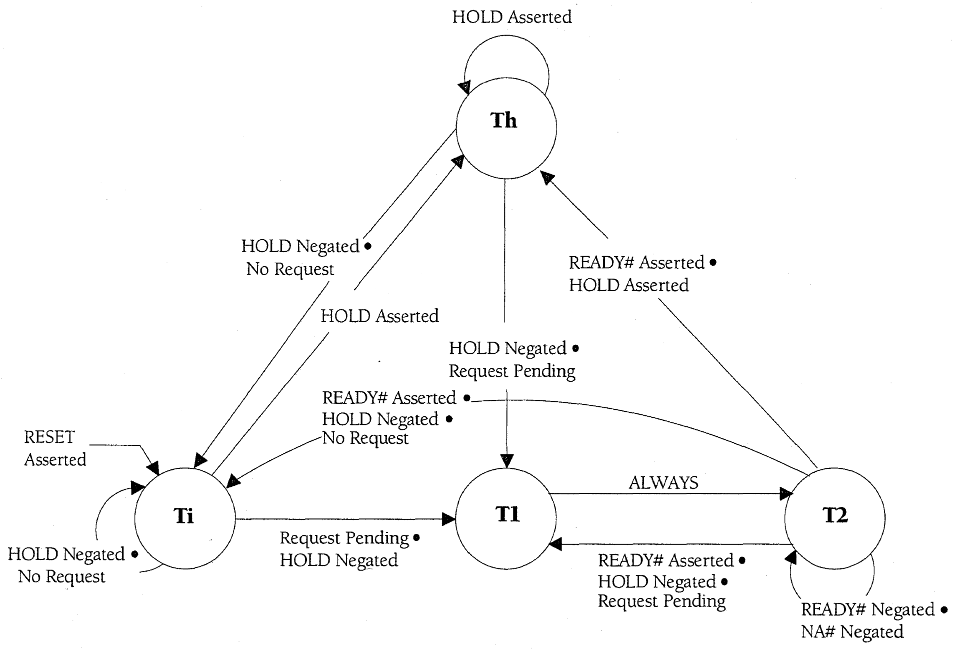


Figure 3-6. Various Non-Pipelined Bus Cycles with Different Numbers of Wait States

Initiating and Maintaining Non-Pipelined Cycles

The bus states and transitions for non-pipelined addressing are illustrated in Figure 3-7. The bus transitions between four possible states: T1, T2, Ti, and Th. Active bus cycles consist of T1 and T2 states, with T2 being 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 is repeated resulting in a wait state. When a cycle is acknowledged during T2, the following state is T1 of the next bus cycle if a bus request is pending internally. If no internal bus request is pending, the Ti state is entered. If the


HOLD Asserted
Th
HOLD Negated • No Request
HOLD Asserted
READY# Asserted • HOLD Asserted
HOLD Negated • Request Pending
READY# Asserted • HOLD Negated • No Request
RESET Asserted
Ti
HOLD Negated • No Request
Request Pending • HOLD Negated
T1
ALWAYS
T2
READY# Asserted • HOLD Negated • Request Pending
READY# Negated • NA# Negated

Bus States:
T1 -- First clock of a non-pipelined 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.

1712600

Figure 3-7. Non-Pipelined Bus States

HOLD input is asserted and the Cx486DLC is ready to enter the hold acknowledge state, the Th state is entered.

Due to the demultiplexed nature of the bus, the address pipelining option provides a mechanism for the external hardware to have an additional T state worth of access time without inserting a wait state. After the reset sequence and following any idle bus state, the processor always uses non-pipelined address timing. Pipelined or non-pipelined 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 it is desirable to maintain non-pipelined address timing, it is necessary to negate NA# during each T2 state of the bus cycle except the last one.

3.3.2.2 Bus Cycles using Pipelined Addressing

The address pipelining option allows the system to 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. The address pipelining option is controlled on a cycle-by-cycle basis by the state of the NA# input.

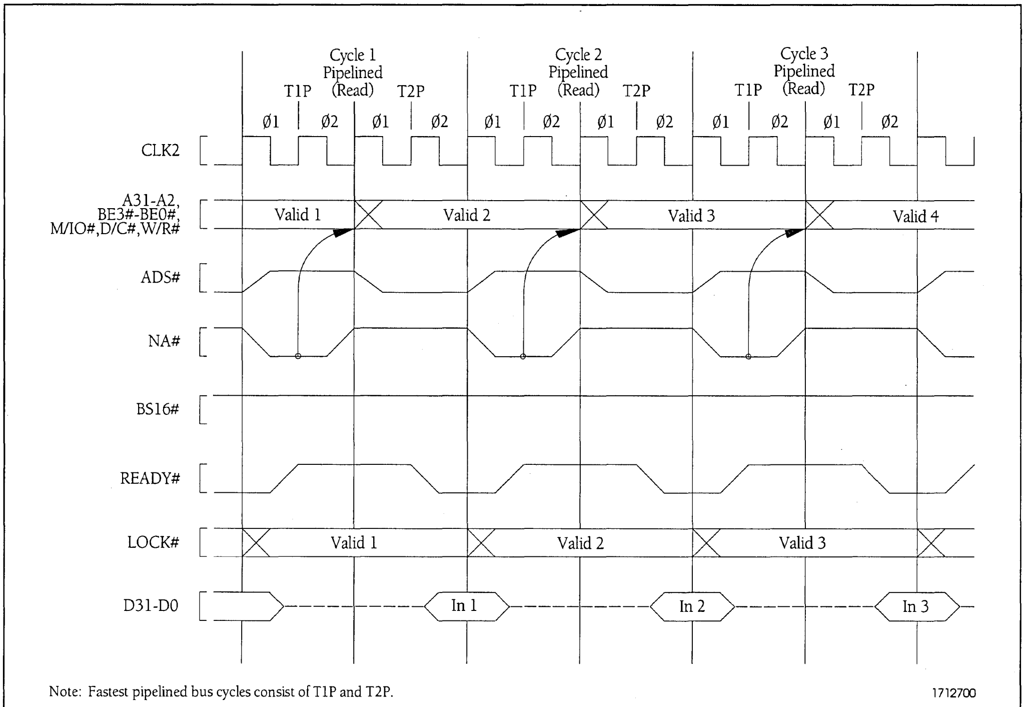
Pipelined Bus States

Pipelined addressing is always initiated by asserting NA# during a non-pipelined bus cycle. Within the non-pipelined bus cycle, NA# is sampled at the beginning of phase 2 of each T2 state and is only acknowledged by the Cx486DLC 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 non-pipelined cycle. The address strobe output (ADS#) is asserted simultaneously to indicate validity of the above signals. Once in effect, address pipelining is maintained in successive bus cycles by continuing to assert NA# during the pipelined bus cycles.

As in non-pipelined bus cycles, the fastest bus cycles using 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 is the first bus state of every pipelined bus cycle.

Within the pipelined bus cycle, NA# is sampled at the beginning of phase 2 of the T1P state. If the Cx486DLC 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 state 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.



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

Figure 3-8. Fastest Pipelined Read Cycles

Pipelined Read and Write Cycles

Any bus cycle may be performed with pipelined address timing. When a read cycle is performed, the Cx486DLC microprocessor floats its data bus and the externally addressed device then drives the data. When a read cycle is acknowledged by READY# asserted in either the T2P or T2I bus state, the Cx486DLC CPU 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 Cx486DLC CPU beginning in phase two of T1P. When a write cycle is acknowledged, the Cx486DLC write data remains valid throughout phase one of the next bus state to provide write data hold time.

Pipelined Wait States

Once a pipelined bus cycle begins, it continues until acknowledged by the external system hardware using the Cx486DLC 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 Cx486DLC ignores the READY# input at the end of the T1P state. Note that 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 effects. Pipelined addressing can only output information for

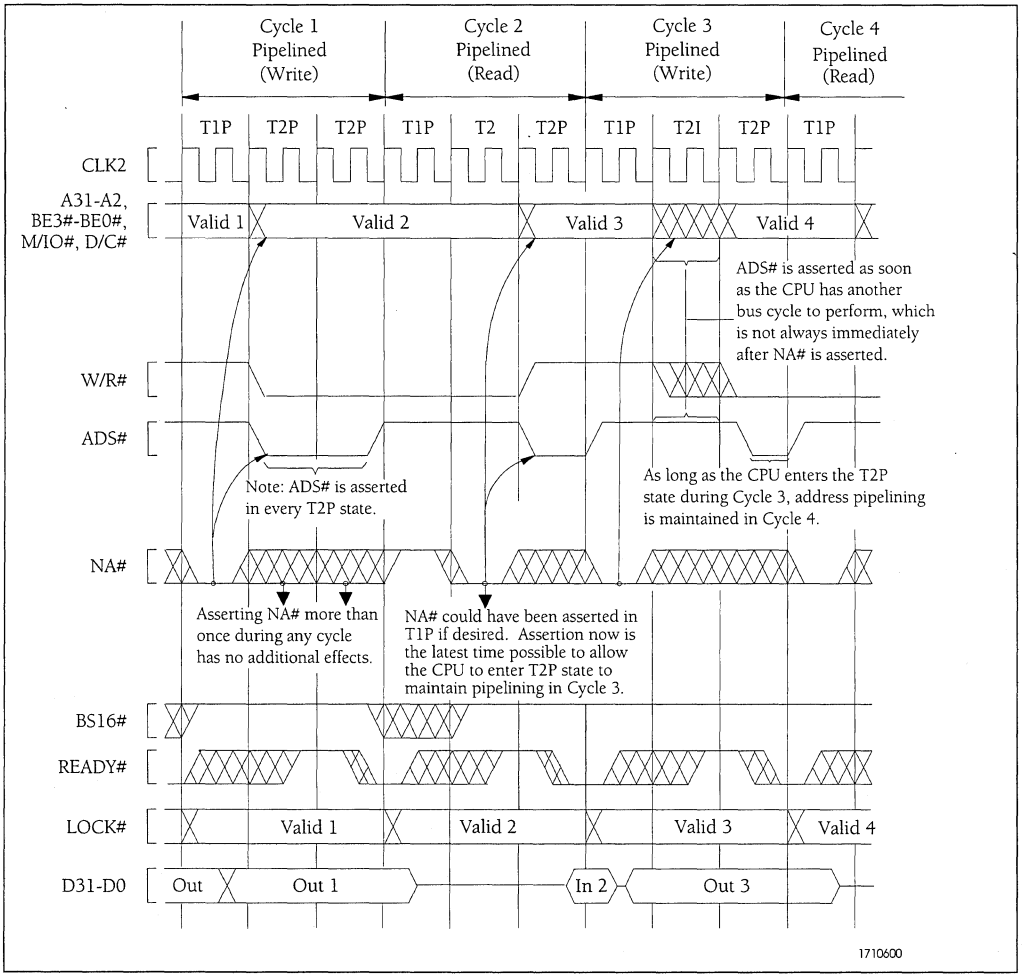

Cycle 1 Pipelined (Write)
Cycle 2 Pipelined (Read)
Cycle 3 Pipelined (Write)
Cycle 4 Pipelined (Read)
T1P
T2P
T2P
T1P
T2
T2P
T1P
T2I
T2P
T1P
CLK2
A31-A2, BE3#-BE0#, M/IO#, D/C#
Valid 1
Valid 2
Valid 3
Valid 4
ADS# is asserted as soon as the CPU has another bus cycle to perform, which is not always immediately after NA# is asserted.
W/R#
ADS#
Note: ADS# is asserted in every T2P state.
As long as the CPU enters the T2P state during Cycle 3, address pipelining is maintained in Cycle 4.
NA#
Asserting NA# more than once during any cycle has no additional effects.
NA# could have been asserted in T1P if desired. Assertion now is the latest time possible to allow the CPU to enter T2P state to maintain pipelining in Cycle 3.
BS16#
READY#
LOCK#
Valid 1
Valid 2
Valid 3
Valid 4
D31-D0
Out
Out 1
In 2
Out 3
1710600

Figure 3-9. Various Pipelined Cycles (one wait state)

the very 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 Cx486DLC samples NA# asserted. Because a bus request is pending internally and READY# is not active, the CPU enters the T2P state and asserts ADS#, 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#, 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# need only be asserted once during the bus cycle to request pipelined addressing.

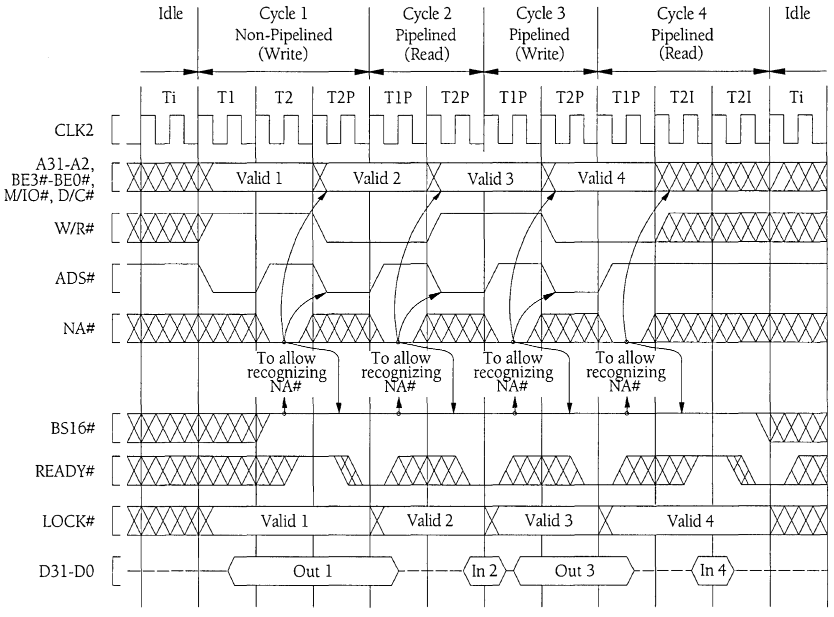
Initiating and Maintaining Pipelined Cycles

Pipelined addressing is always initiated by asserting NA# during a non-pipelined bus cycle with at least one wait state. The first bus cycle following RESET, an idle bus or a hold acknowledge state is always non-pipelined. Therefore, the Cx486DLC always issues at least one non-pipelined 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 Cx486DLC 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.

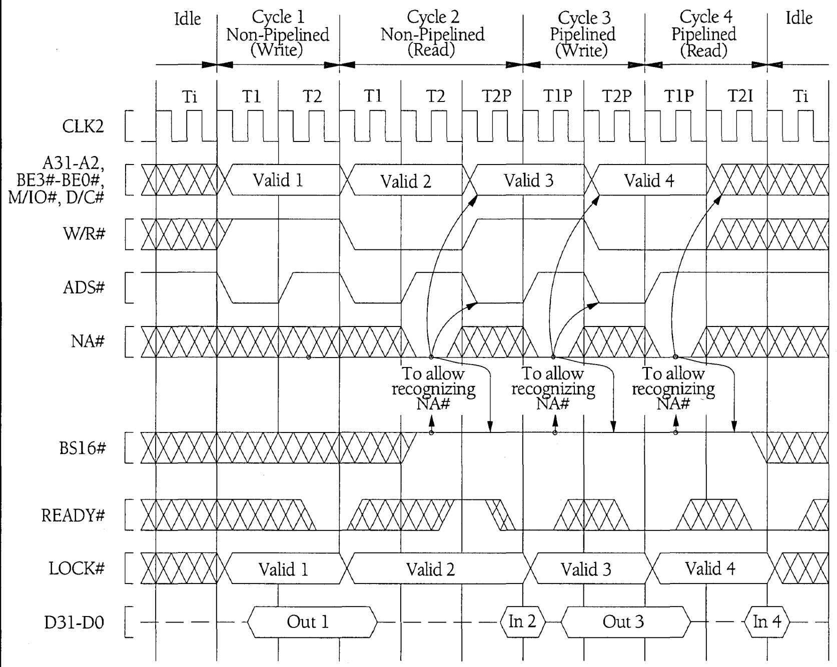
Figure 3-10 illustrates the fastest transition possible to pipelined addressing following an idle bus state. In Cycle 1, the next address is driven during state T2P. Thus, Cycle 1 makes the transition to pipelined address timing, since it begins with T1 but ends with T2P. Because the address for Cycle 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# asserted terminates Cycle 1.



Note: Following any idle bus state (Ti), the address is always non-pipelined and NA# is only sampled during wait states. To start address pipelining after an idle state requires a non-pipelined cycle with at least one wait state (Cycle 1 above). The pipelined cycles (2, 3, and 4 above) are shown with various numbers of wait states. 1710700

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

Figure 3-11 illustrates transitioning 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 illustrates that a transition cycle is the same whenever it occurs consisting of at least T1, T2 (NA# is asserted at that time), and T2P (provided the Cx486DLC 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.



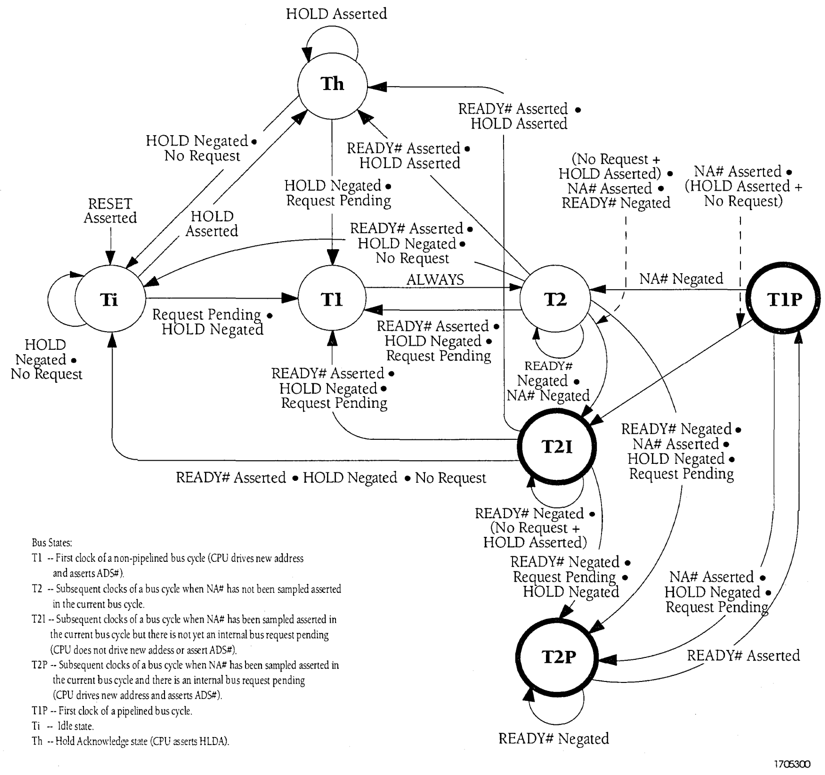
Note: Following any idle bus state (Ti), addresses are non-pipelined. Within non-pipelined bus ycles, NA# is only sampled during wait states. Therefore, to begin address pipelining during a group of non-pipelined bus cycles requires a non-pipelined cycle with at least one wait state (Cycle 2 above).

1710800

Figure 3-11. Transitioning to Pipelined Address During Burst of Bus Cycles

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting NA# and detecting that the Cx486DLC 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#. Figures 3-10 and 3-11 each show pipelining ending after Cycle 4. This occurred because the Cx486DLC CPU did not have an internal bus request prior to the acknowledgment of Cycle 4.

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 non-pipelined address. The three additional bus states for pipelined address are shown in heavier circles.


HOLD Asserted
Th
READY# Asserted • HOLD Asserted
HOLD Negated • No Request
READY# Asserted • HOLD Asserted
HOLD Negated • Request Pending
(No Request + HOLD Asserted) • NA# Asserted • READY# Negated
NA# Asserted • (HOLD Asserted + No Request)
RESET Asserted
HOLD Asserted
READY# Asserted • HOLD Negated • No Request
ALWAYS
NA# Negated
Ti
T1
T2
T1P
Request Pending • HOLD Negated
READY# Asserted • HOLD Negated • Request Pending
HOLD Negated • No Request
READY# Asserted • HOLD Negated • Request Pending
READY# Negated • NA# Negated
T2I
READY# Negated • NA# Asserted • HOLD Negated • Request Pending
READY# Asserted • HOLD Negated • No Request
READY# Negated • (No Request + HOLD Asserted)
READY# Negated • Request Pending • HOLD Negated
NA# Asserted • HOLD Negated • Request Pending
T2P
READY# Asserted
READY# Negated
1705300

Bus States:
T1 -- First clock of a non-pipelined 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 new addess 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).

Figure 3-12. Complete Bus States

3.3.2.3 Bus Cycles Using BS16#

Assertion of BS16# during a bus cycle effectively changes the Cx486DLC 32-bit data 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.

Non-Pipelined Cycles

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

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

For data writes with only the two upper bytes selected (BE3# and/or 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#, BE2# asserted and perhaps BE0# and/or 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#, BE2# asserted and perhaps BE0# and/or 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 and bytes 2 and 3 (corresponding to BE2# and BE3#) are sent on the second bus cycle. BE0# and BE1# are always negated during the second 16-bit bus cycle. Figure 3-13 illustrates two non-pipelined bus cycles using BS16#.

Pipelined Cycles

The input signal "NA#" is a request to the CPU to drive the address, byte enables and 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 sampled asserted in the same window, NA# is ignored and BS16# remains effective (that is the data bus becomes 16-bits wide). Figure 3-14 illustrates the interaction between NA# and BS16#.

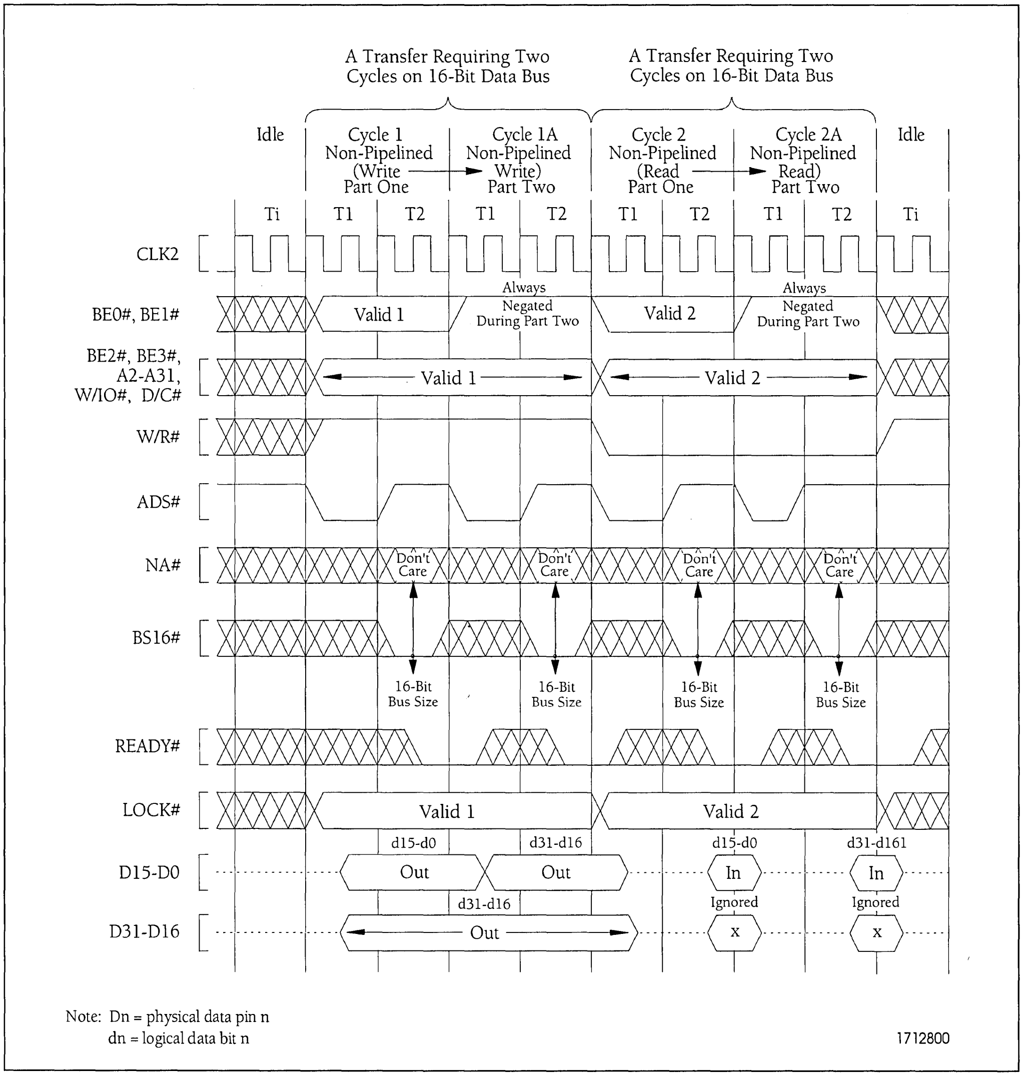
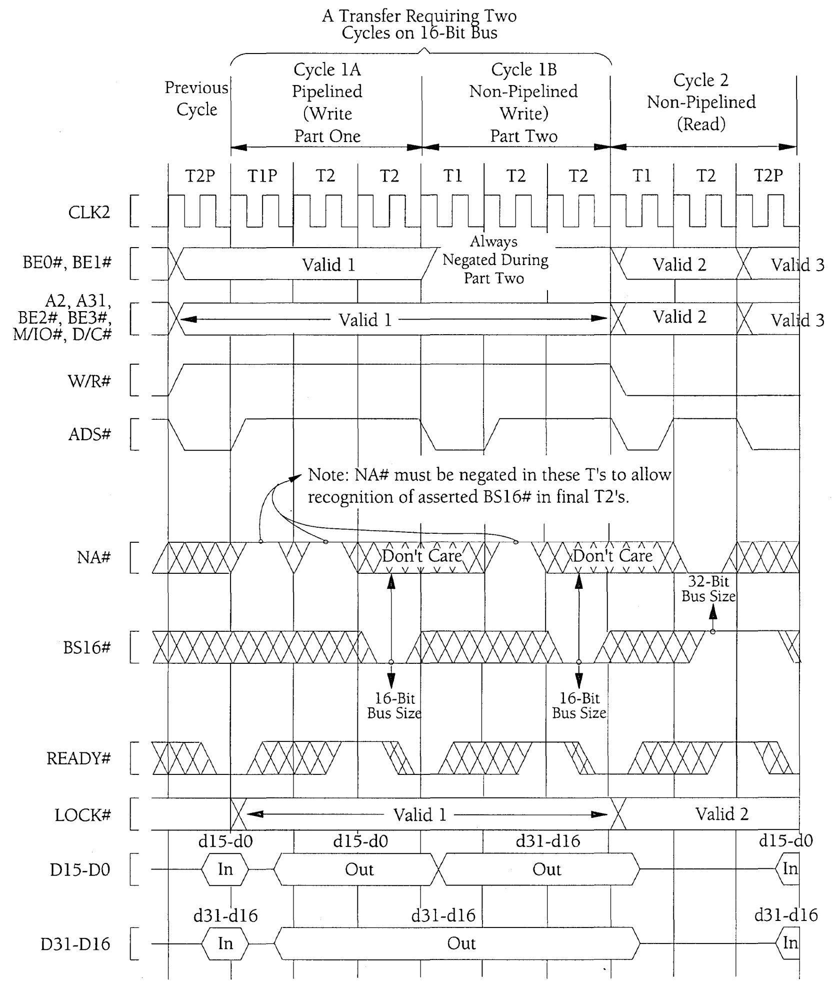

A Transfer Requiring Two Cycles on 16-Bit Data Bus
A Transfer Requiring Two Cycles on 16-Bit Data Bus
Idle
Cycle 1 Non-Pipelined (Write Part One
Cycle 1A Non-Pipelined Write) Part Two
Cycle 2 Non-Pipelined (Read Part One
Cycle 2A Non-Pipelined Read) Part Two
Idle
Ti
T1
T2
CLK2
BE0#, BE1#
Valid 1
Always Negated During Part Two
Valid 2
BE2#, BE3#, A2-A31, W/IO#, D/C#
W/R#
ADS#
NA#
Don't Care
BS16#
16-Bit Bus Size
READY#
LOCK#
D15-D0
d15-d0
d31-d16
Out
In
D31-D16
Ignored
x
Note: Dn = physical data pin n
dn = logical data bit n
1712800

Figure 3-13. Non-Pipelined Bus Cycles Using BS16#


A Transfer Requiring Two Cycles on 16-Bit Bus
Previous Cycle
Cycle 1A Pipelined (Write Part One
Cycle 1B Non-Pipelined Write) Part Two
Cycle 2 Non-Pipelined (Read)
T2P
T1P
T2
T2
T1
T2
T2
T1
T2
T2P
CLK2
BE0#, BE1#
Valid 1
Always Negated During Part Two
Valid 2
Valid 3
A2, A31, BE2#, BE3#, M/IO#, D/C#
Valid 1
Valid 2
Valid 3
W/R#
ADS#
Note: NA# must be negated in these T's to allow recognition of asserted BS16# in final T2's.
NA#
Don't Care
Don't Care
32-Bit Bus Size
BS16#
16-Bit Bus Size
16-Bit Bus Size
READY#
LOCK#
Valid 1
Valid 2
d15-d0
d15-d0
d31-d16
d15-d0
D15-D0
In
Out
Out
In
d31-d16
d31-d16
d31-d16
D31-D16
In
Out
In

Dn = physcial 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 externally simulate address pipelining during cycle 1B.

1711900

Figure 3-14. Pipelining and BS16#

3.3.3 Locked Bus Cycles

When the LOCK# signal is asserted, the Cx486DLC microprocessor does 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 (described next).

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 using non-pipelined addressing, LOCK# is asserted during phase 1 of T1. When using pipelined addressing, LOCK# is driven valid during phase 1 of T1P.

Figures 3-4 through 3-6 illustrate LOCK# timing during non-pipelined cycles and Figures 3-8 through 3-11 cover the pipelined address case.

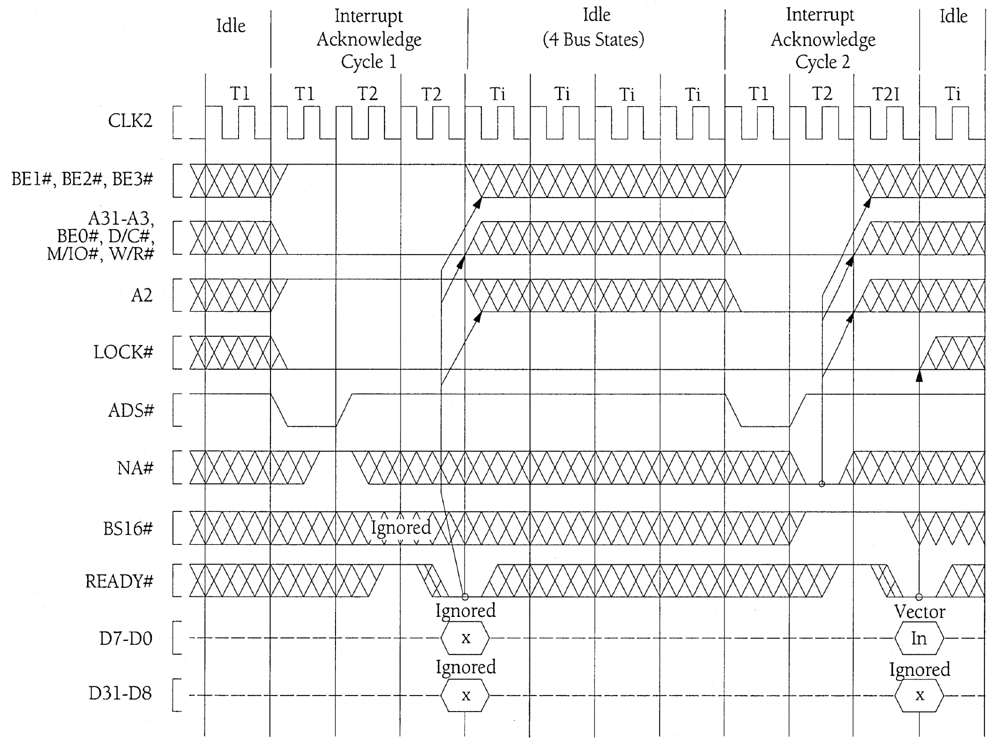
3.3.4 Interrupt Acknowledge (INTA) Cycles

The Cx486DLC microprocessor is interrupted by an external source via an input request on the INTR input (when interrupts are enabled). The Cx486DLC 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 3-15.

The state of A2 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. Four idle bus states (Ti) are always inserted by the Cx486DLC microprocessor between the two interrupt acknowledge cycles.

The interrupt vector is read at the end of the second interrupt cycle. The vector is read by the Cx486DLC microprocessor 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 Cx486DLC is ignored.



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.

1710900

Figure 3-15. Interrupt Acknowledge Cycles

3.3.5 Halt and Shutdown Cycles

Halt Indication Cycle

Executing the HLT instruction causes the Cx486DLC execution unit to cease operation. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the state of the bus cycle definition signals (M/IO#=1, D/C#=0, W/R#=1, LOCK#=1) and an address of 2h (A31-A2=0, BE3#=1, BE2#=0, BE1#-BE0#=1). The halt indication cycle must be acknowledged by READY# asserted. A halted Cx486DLC microprocessor resumes execution when INTR (if interrupts are enabled), NMI, or RESET is asserted. Figure 3-16 illustrates a non-pipelined halt cycle.

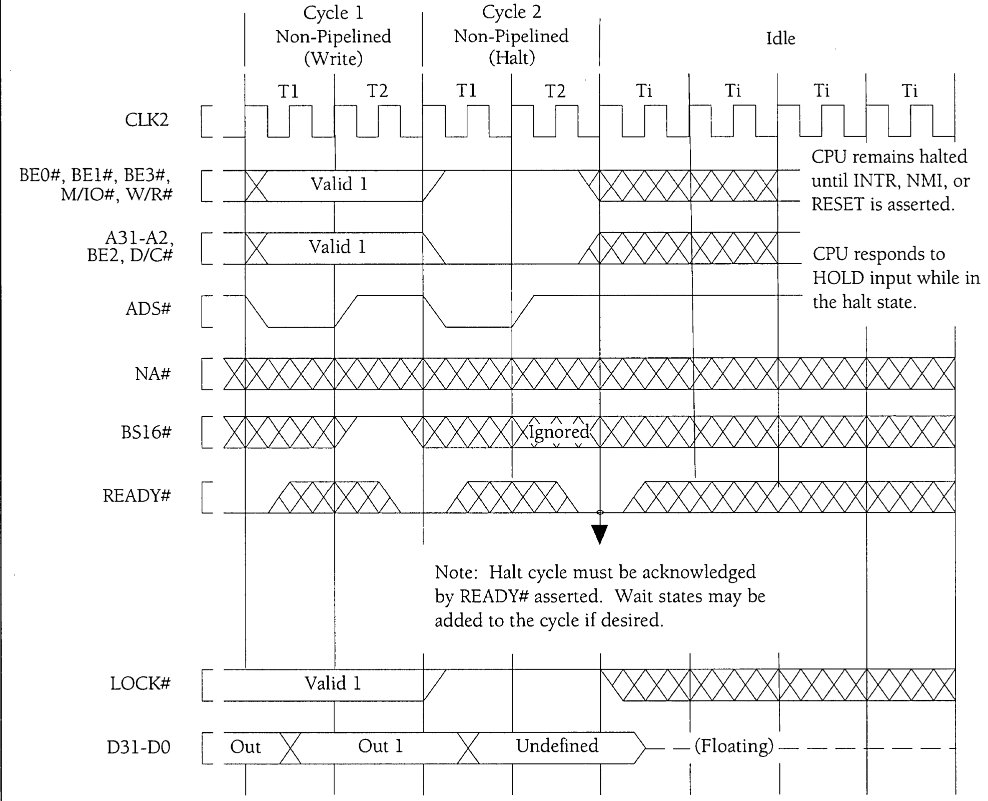

Cycle 1
Non-Pipelined
(Write)
Cycle 2
Non-Pipelined
(Halt)
Idle
T1
T2
T1
T2
Ti
Ti
Ti
Ti
CLK2
BE0#, BE1#, BE3#, M/IO#, W/R#
Valid 1
CPU remains halted until INTR, NMI, or RESET is asserted.
A31-A2, BE2, D/C#
Valid 1
CPU responds to HOLD input while in the halt state.
ADS#
NA#
BS16#
Ignored
READY#
Note: Halt cycle must be acknowledged by READY# asserted. Wait states may be added to the cycle if desired.
LOCK#
Valid 1
D31-D0
Out
Out 1
Undefined
(Floating)
1711000

Figure 3-16. Non-pipelined Halt Cycle

Shutdown Indication Cycle

Shutdown occurs when a severe error is detected that prevents further processing. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the state of the bus cycle definition signals (M/IO#=1, D/C#=0, W/R#=1, LOCK=1) and an address of 0h (A31-A1 = 0, BE3#-BE1#=1, BE0#=0). The shutdown indication cycle must be acknowledged by READY# asserted. A shut down Cx486DLC microprocessor resumes execution only when NMI or RESET is asserted. Figure 3-17 illustrates a shutdown cycle using pipelined addressing.

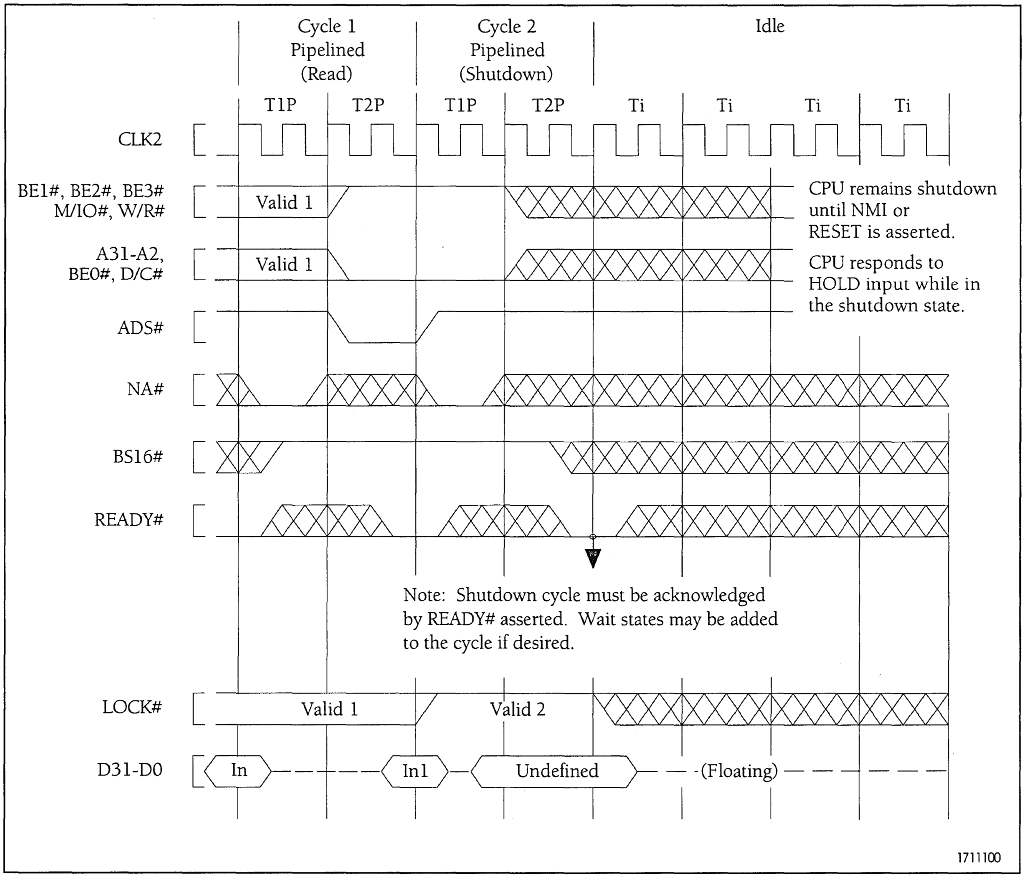

Cycle 1 Pipelined (Read)
Cycle 2 Pipelined (Shutdown)
Idle
T1P
T2P
T1P
T2P
Ti
Ti
Ti
Ti
CLK2
BE1#, BE2#, BE3# M/IO#, W/R#
Valid 1
CPU remains shutdown until NMI or RESET is asserted.
A31-A2, BE0#, D/C#
Valid 1
CPU responds to HOLD input while in the shutdown state.
ADS#
NA#
BS16#
READY#
Note: Shutdown cycle must be acknowledged by READY# asserted. Wait states may be added to the cycle if desired.
LOCK#
Valid 1
Valid 2
D31-D0
In
In1
Undefined
(Floating)
1711100

Figure 3-17. Pipelined Shutdown Cycle

3.3.6 Internal Cache Interface

3.3.6.1 Cache Fills

Any unlocked memory read cycle can be cached by the Cx486DLC. The Cx486DLC automatically does not cache accesses to memory addresses specified by the non-cacheable region registers. Additionally, the KEN# input can be used to enable caching of memory accesses on a cycle-by-cycle basis. The Cx486DLC acknowledges the KEN# input only if the KEN enable bit is set in the CCR0 configuration register.

As shown in Figures 3-18 and 3-19, the Cx486DLC samples the KEN# input one CLK2 before READY# is sampled active. If KEN# is asserted and the current address is not set as non-cacheable per the non-cacheable region registers, then the Cx486DLC fills four 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.

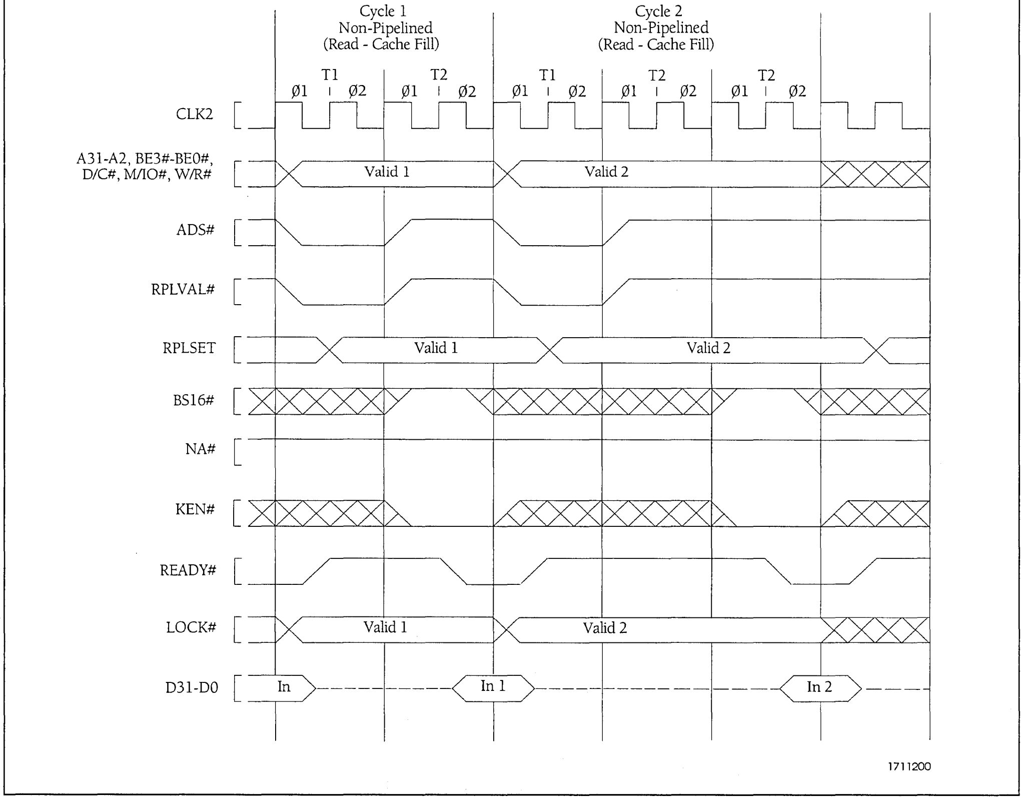


Figure 3-18. Non-Pipelined Cache Fills using KEN#

If the RPL bit in the CCR1 configuration register is set, then the RPLSET and RPLVAL# output signals are driven by the Cx486DLC during cache fill cycles. RPLSET is only meaningful if the cache is configured as two-way set associative. RPLSET indicates which set in the cache is undergoing a line replacement. RPLVAL# indicates that the Cx486DLC will perform a cache fill to the indicated set with the data present on the data bus pins at the time READY# is sampled active.

However, if KEN# is enabled and sampled inactive, the data is not cached and the line in the set indicated by RPLSET is not overwritten.

Figures 3-18 and 3-19 illustrate RPLVAL# and RPLSET functional timing for non-pipelined cache fills and pipelined cache fills respectively. RPLVAL# is driven at the same time and for the same duration as the ADS# output for the cache fill cycle. RPLSET is driven one CLK2 after

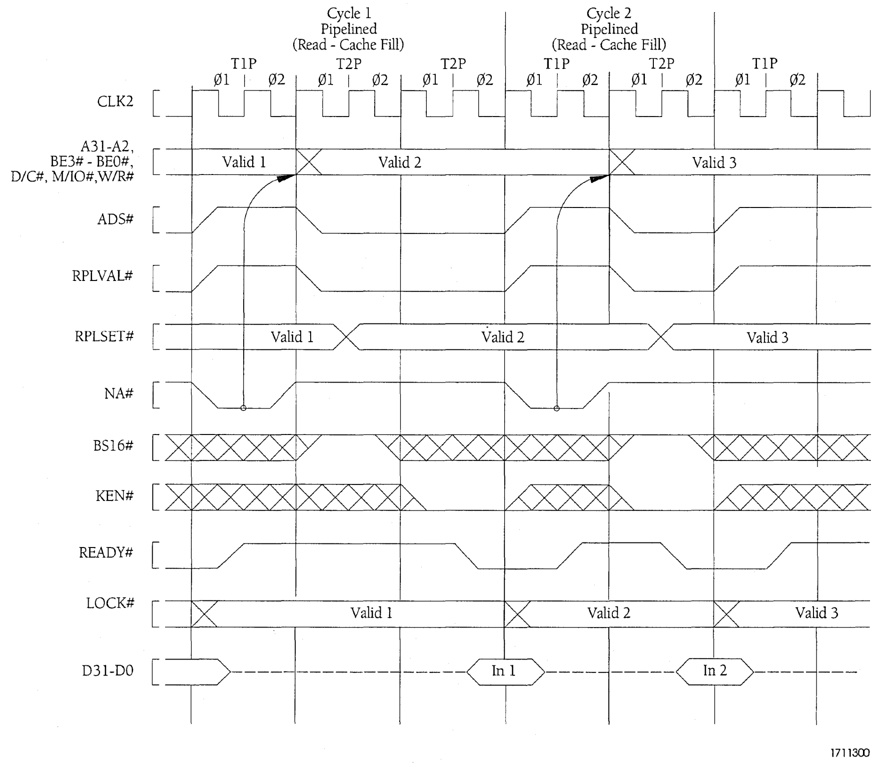


Figure 3-19. Pipelined Cache Fills using KEN#

RPLVAL# is driven regardless of whether or not the current bus cycle is pipelined.

If BS16# is asserted along with KEN# during a read operation, the Cx486SLC may perform two consecutive reads to complete a cache line fill. Each read operation with BS16# asserted provides two bytes of data on D15-D0 to fill the 4-byte cache line. KEN# must be asserted along with BS16# for both cycles in order for the cache fill to occur. This is illustrated in Figure 3-20.

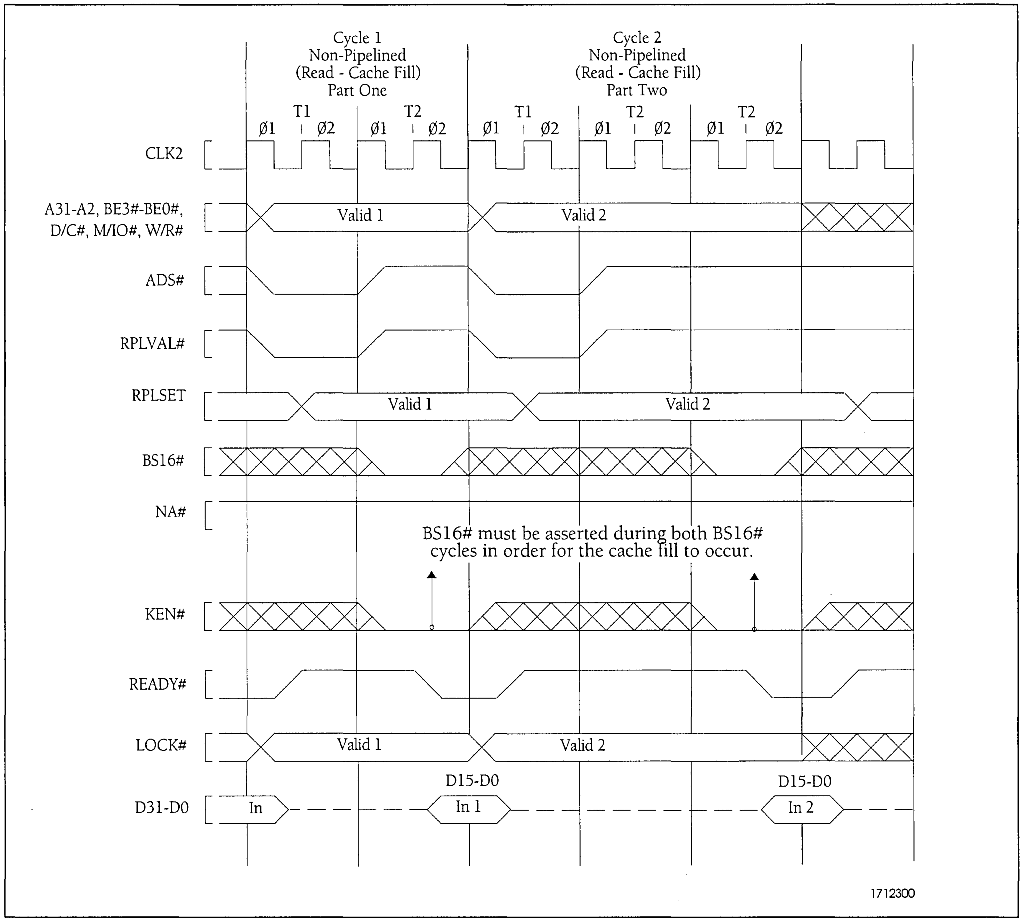


Figure 3-20. Non-Pipelined Cache Fills using KEN# and BS16#

3.3.6.2 Flushing the Cache

To maintain cache coherency with external memory, the Cx486DLC cache contents should be invalidated when previously cached data is modified in external memory by another bus master. The Cx486DLC 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, or following assertion of FLUSH# if the FLUSH bit is set in CCR0.

The Cx486DLC samples the FLUSH# input on the rising edge of CLK2 corresponding to the beginning of phase 2 of the internal processor clock. If FLUSH# is asserted, the Cx486DLC invalidates the entire contents of the internal cache. The actual point in time where 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.

For maximum performance, FLUSH# should only be asserted for the minimum time reguired. Each phase 2 edge where FLUSH# is sampled asserted causes the CPU to slow execution and invalidate the cache contents. The CPU invalidates the cache even if a cache invalidation had just occurred during the previous clock cycle.

3.3.7 Address Bit 20 Masking

The Cx486DLC internal cache addressing must be forced to emulate 8086 1 MByte wrap-around addressing when system logic emulates the wrap-around addressing and data within the 64 KByte wrap-around area resides in the Cx486DLC internal cache. The Cx486DLC emulates the wrap-around addressing if the A20 bit is set in the CCR0 configuration register and the A20M# input is asserted. Both the address bit 20 input to the internal cache and the external A20 pin are masked (zeroed) when the A20M# input is asserted.

As shown in Figure 3-21, the Cx486DLC samples the A20M# input on the rising edge of CLK2 corresponding to the beginning of phase 2 of the internal processor clock. If A20M# is asserted and paging is not enabled, the Cx486DLC 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#. A20 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.

An alternative to using the A20M# pin is provided by the NC0 bit in the CCR0 configuration register. The Cx486DLC automatically does not cache accesses to the first 64 KBytes and to the first 64 KBytes at each 1 MByte boundary if the NC0 bit is set. This prevents data within the wrap-around memory area from residing in the internal cache and thus eliminates the need for masking A20 to the internal cache.

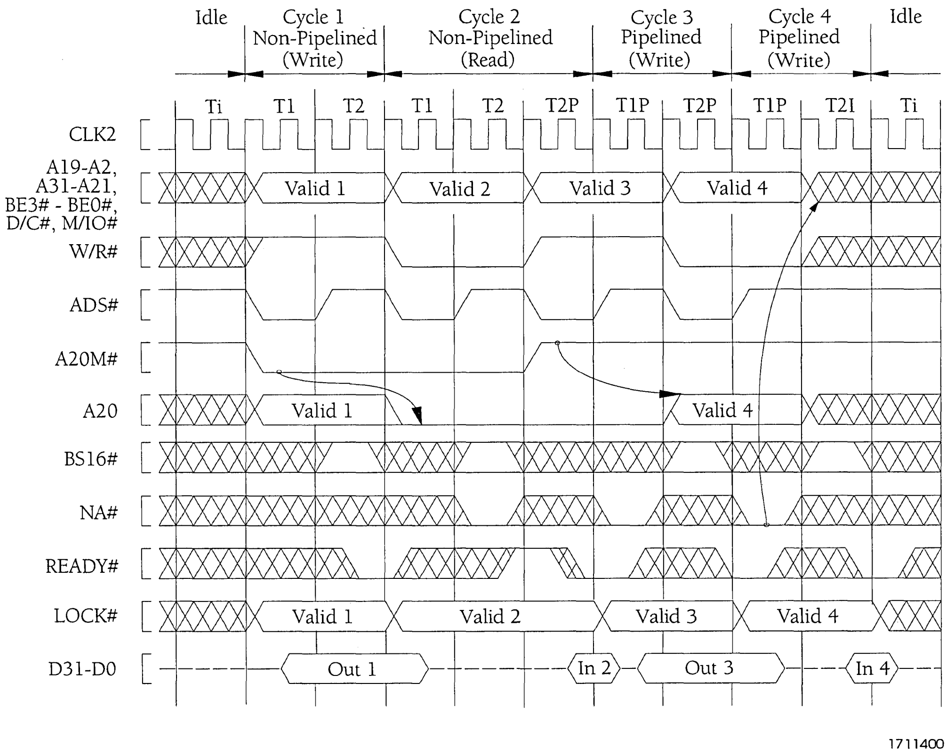


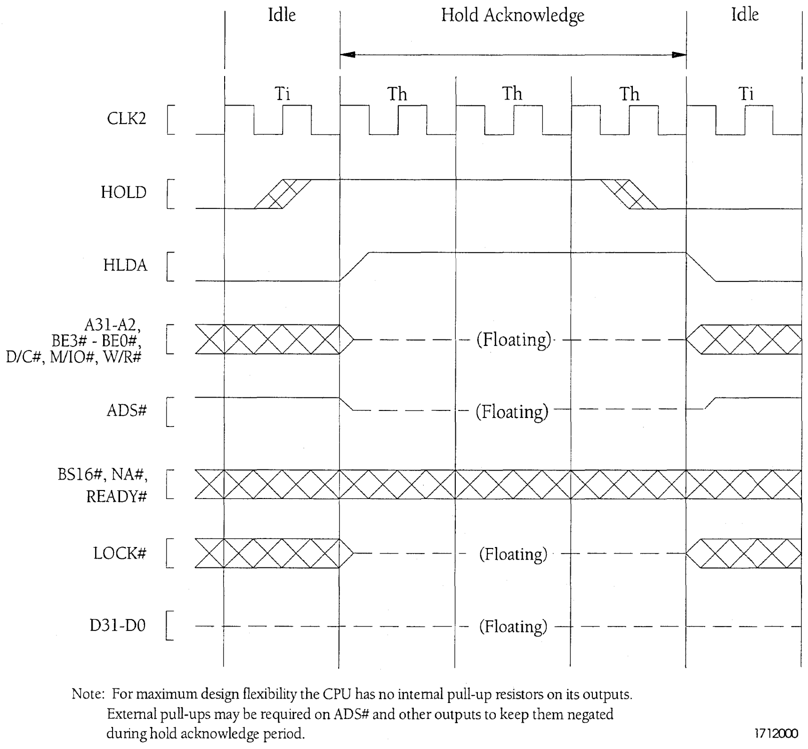
Figure 3-21. Masking A20 using A20M# During Burst of Bus Cycles

3.3.8 Hold Acknowledge State

The hold acknowledge state provides the mechanism for an external device in a Cx486DLC system to acquire the Cx486DLC system bus while the Cx486DLC is held in an inactive bus state. This allows external "bus masters" to take control of the Cx486DLC bus and directly access system hardware in a shared manner with the Cx486DLC. The Cx486DLC continues to execute instructions out of the 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 Cx486DLC microprocessor floats all output and bi-directional signals, except for HLDA and SUSPA#. HLDA is asserted as long as the Cx486DLC CPU remains in the hold acknowledge state and all inputs except HOLD, FLUSH#, SUSP# and RESET are ignored.

Th may be entered directly from a bus idle state, as in Figure 3-22, or after the completion of the current physical bus cycle if the LOCK# signal is

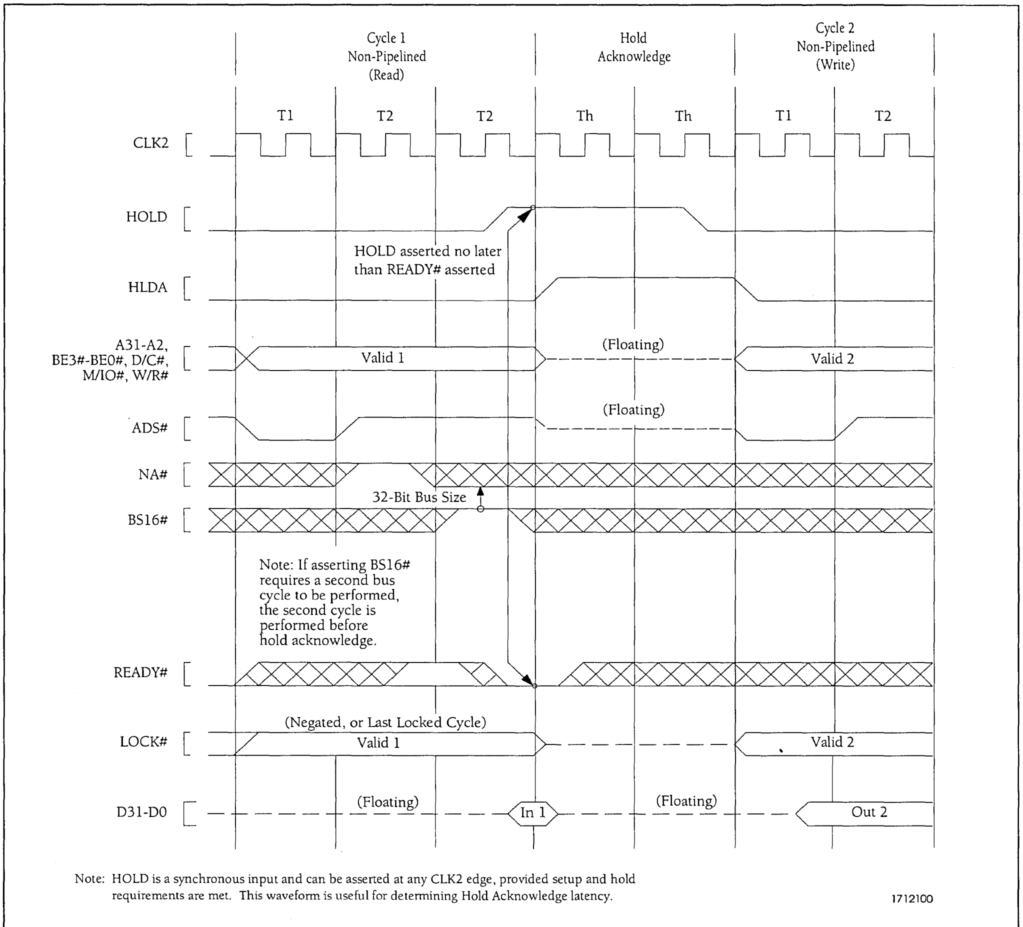


Idle
Hold Acknowledge
Idle
Ti
Th
Th
Th
Ti
CLK2
HOLD
HLDA
A31-A2, BE3# - BE0#, D/C#, M/IO#, W/R#
(Floating)
ADS#
(Floating)
BS16#, NA#, READY#
LOCK#
(Floating)
D31-D0
(Floating)
Note: For maximum design flexibility the CPU has no internal pull-up resistors on its outputs. External pull-ups may be required on ADS# and other outputs to keep them negated during hold acknowledge period.
1712000

Figure 3-22. Requesting Hold from Idle Bus State

not asserted, as in Figures 3-23 and 3-24. The CPU samples the HOLD input on the rising edge of CLK2 corresponding to the beginning of phase 1 of the internal processor clock. HOLD must meet specified setup and hold times to be recognized at a given CLK2 edge.

The hold acknowledge state is exited in response to the HOLD input being negated. The next bus state is an idle state (Ti) if no bus request is pending, as in Figure 3-22. If a bus request is internally pending, as in Figures 3-23 and 3-24, the next bus state is T1. Th is also exited in response to RESET being asserted. If HOLD

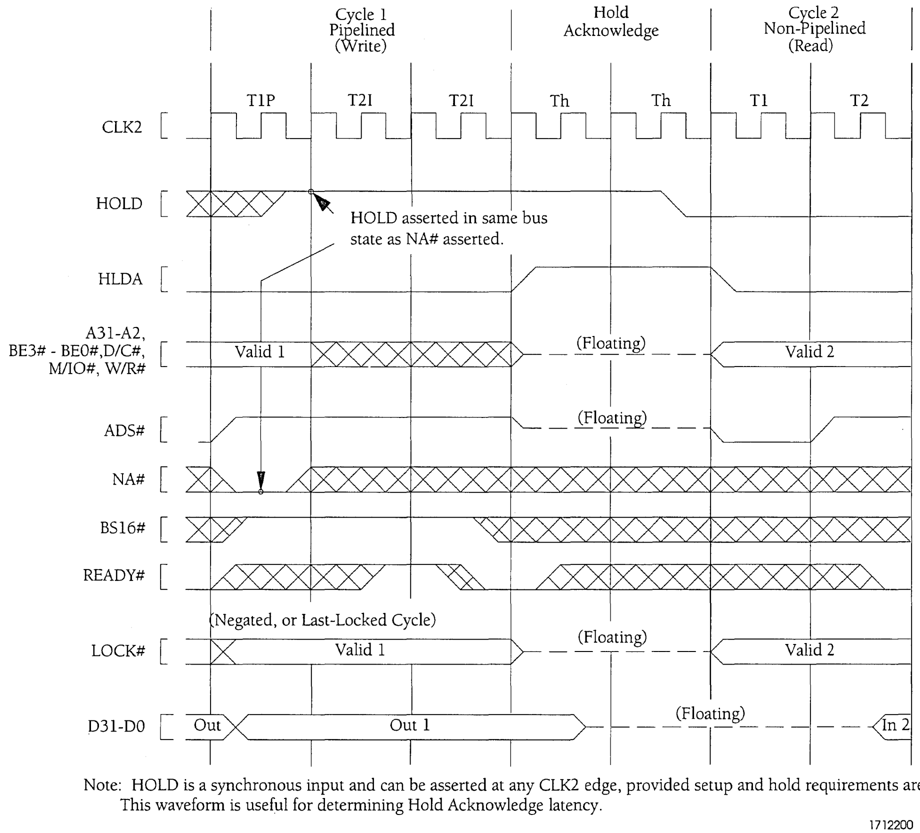

Cycle 1
Non-Pipelined
(Read)
Hold
Acknowledge
Cycle 2
Non-Pipelined
(Write)
T1
T2
T2
Th
Th
T1
T2
CLK2
HOLD
HOLD asserted no later than READY# asserted
HLDA
A31-A2, BE3#-BE0#, D/C#, M/IO#, W/R#
Valid 1
(Floating)
Valid 2
ADS#
(Floating)
NA#
32-Bit Bus Size
BS16#
Note: If asserting BS16# requires a second bus cycle to be performed, the second cycle is performed before hold acknowledge.
READY#
(Negated, or Last Locked Cycle)
LOCK#
Valid 1
Valid 2
D31-D0
(Floating)
In 1
(Floating)
Out 2
1712100

Note: HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 3-23. Requesting Hold from Active Non-Pipelined Bus

remains asserted when RESET goes inactive, the Cx486DLC 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 the Th state, the event is remembered as a non-maskable interrupt 2 and is serviced when the Th state is exited.



Note: HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 3-24. Requesting Hold from Active Pipelined Bus

3.3.9 Coprocessor Interface

The coprocessor interface consists of the data bus, address bus, bus cycle definition signals, and the coprocessor interface signals (BUSY#, ERROR# and PEREQ). The Cx486DLC automatically accesses dedicated coprocessor I/O addresses 8000 00F8h, and 8000 00FCh to transfer opcodes and operands to/from the coprocessor whenever a coprocessor instruction is decoded. Coprocessor cycles can be either read or write and can be either non-pipelined 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 used to synchronize CPU and coprocessor operation. All three signals are sampled at the beginning of phase 1 and must meet specified setup and hold times to be recognized at a given CLK2 edge.

3.3.10 Power Management

SUSP# Initiated Suspend Mode

The Cx486DLC 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 Cx486DLC also waits for the coprocessor to indicate a not busy status (BUSY# = 1) prior to entering suspend mode. The SUSPA# output is then asserted. The Cx486DLC responds to SUSP# and asserts SUSPA# only if the SUSP bit is set in the CCR0 configuration register.

Figure 3-25 illustrates the Cx486DLC functional timing for SUSP# initiated suspend mode. SUSP# is sampled on the phase 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 2 CLK2s. As a maximum, the Cx486DLC may execute up to two instructions and associated bus cycles prior to asserting SUSPA#. The time required for the Cx486DLC to deactivate SUSPA# once SUSP# has been sampled inactive is 4 CLK2s.

If the Cx486DLC 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 Cx486DLC internal execution pipeline. If the Cx486DLC is in a SUSP# initiated suspend state and the CLK2 input is not stopped, the processor recognizes and acknowledges the HOLD input and stores the occurrence of FLUSH#, NMI and INTR (if enabled) for execution once suspend mode is exited.

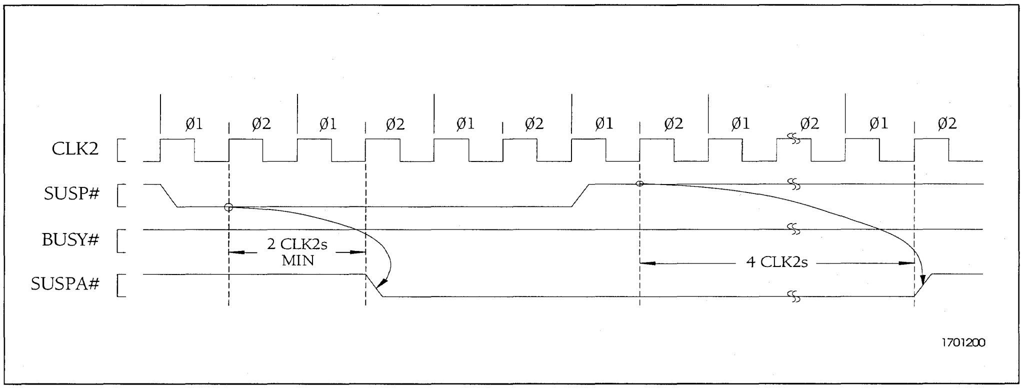


Figure 3-25. SUSP# Initiated Suspend Mode

Halt Initiated Suspend Mode

The Cx486DLC also enters suspend mode as a result of executing a HLT instruction. The SUSPA# output is asserted no more than 17 CLK2s following READY# sampled active for the halt bus cycle as shown in Figure 3-26. Suspend mode is then exited upon recognition of an NMI or an unmasked INTR. SUSPA# is deactivated 12 CLK2s after sampling of an active NMI or unmasked INTR. If the Cx486DLC is in a halt initiated suspend mode and the CLK2 input is not stopped, the processor recognizes and acknowledges the HOLD input and stores the occurrence of FLUSH# for execution once suspend mode is exited.

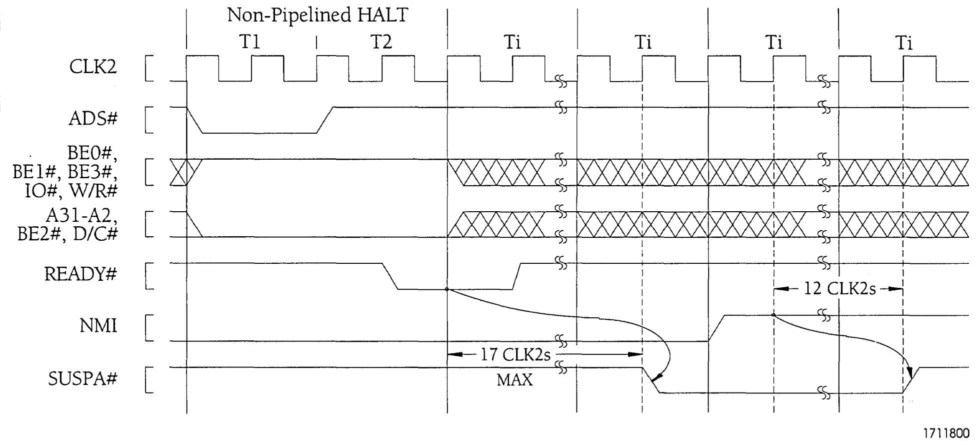


Figure 3-26. Halt Initiated Suspend Mode

Stopping the Input Clock

Because the Cx486DLC is a static device, the input clock (CLK2) can be stopped and restarted without loss of any internal CPU data. CLK2 can be stopped in either phase 1 or phase 2 of the clock and either in a logic high or logic low state. However, entering suspend mode prior to stopping CLK2 dramatically reduces the CPU current requirements. Therefore, the recommended sequence for stopping CLK2 is to initiate Cx486DLC suspend mode, wait for assertion of SUSPA# by the processor and then stop the input clock.

The Cx486DLC remains suspended until CLK2 is restarted and suspend mode is exited as described above. While CLK2 is stopped, the Cx486DLC can no longer sample and respond to any input stimulus including the HOLD, FLUSH#, NMI, INTR and RESET inputs.

Figure 3-27 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 SUSPA# is deasserted.

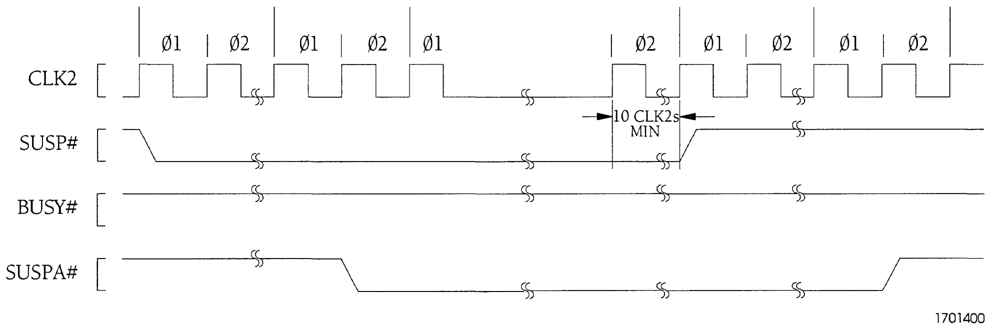


Figure 3-27. Stopping CLK2 During Suspend Mode

4. ELECTRICAL SPECIFICATIONS

4.1 Electrical Connections

4.1.1 Power and Ground Connections and Decoupling

Due to the high frequency of operation of the Cx486DLC, it is necessary to install and test this device using standard high frequency techniques. The high clock frequencies used in the Cx486DLC and its output buffer circuits can cause transient power surges when several output buffers switch output levels simultaneously. These effects can be minimized by filtering the DC power leads with low-inductance decoupling capacitors, using low impedance wiring, and by utilizing all of the 20 V_{CC} and 21 GND pins.

4.1.2 Pull-Up/Pull-Down Resistors

Table 4-1 lists the input pins which are internally connected to pull-up and pull-down resistors. The pull-up resistors are connected to V_{CC} and the pull-down resistors are connected to V_{SS}. When unused, these inputs do not require connection to external pull-up or pull-down resistors.

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

SIGNAL	PIN	RESISTOR
A20M#	F13	20-kΩ pull-up
BS16#	C14	20-kΩ pull-up
BUSY#	B9	20-kΩ pull-up
ERROR#	A8	20-kΩ pull-up
FLUSH#	E13	20-kΩ pull-up
KEN#	B12	20-kΩ pull-up
PEREQ	C8	20-kΩ pull-down
SUSP#	A4	20-kΩ pull-up

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

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

SIGNAL	PIN	EXTERNAL RESISTOR
ADS#	E14	20-kΩ pull-up
LOCK#	C10	20-kΩ pull-up

4.1.3 Unused Input Pins

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

4.1.4 N/C Designated Pins

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

4.2 Absolute Maximum Ratings

The following table lists absolute maximum ratings for the Cx486DLC microprocessor. Stresses beyond those listed in Table 4-3 may cause permanent damage to the device. These are stress ratings only and do not imply that operation under any conditions other than those listed under "Recommended Operating Conditions" (Table 4-4) is possible. Exposure to conditions beyond Table 4-3 may (1) reduce device reliability and (2) result in premature failure even when there is no immediately apparent sign of failure. Prolonged exposure to conditions at or near the absolute maximum ratings (Table 4-3) may also result in reduced useful life and reliability.

Table 4-3. Absolute Maximum Ratings

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

4.3 Recommended Operating Conditions

The following table presents the recommended operating conditions for the Cx486DLC.

Table 4-4. Recommended Operating Conditions

PARAMETER		Cx486DLC		UNITS	NOTES
		MIN	MAX		
T_C	Case Temperature	0°	+85°	C	Power Applied
V_{CC}	Supply Voltage	4.75	5.25	V	With Respect to V_{SS}
V_{IH}	High Level Input	2.0	V_{CC} + 0.3	V	
V_{IL}	Low Level Input	-0.3	0.8	V	
V_{ILC}	CLK2 Input LOW Voltage	-0.3	0.8	V	
V_{IHC}	CLK2 Input HIGH Voltage	3.7	V_{CC} + 0.3	V	
I_{OH}	Output Current (High)		-1.0	mA	$V_{OH} = V_{OH(min)}$
I_{OL}	Output Current (Low)		5.0	mA	$V_{OL} = V_{OL(max)}$
I_{IK}	Input Clamp Current		+10	mA	$V_{IN} < V_{SS}$ or $V_{IN} > V_{CC}$
I_{OK}	Output Clamp Current		+25	mA	$V_{OUT} < V_{SS}$ or $V_{OUT} > V_{CC}$

4.4 DC Characteristics

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

PARAMETER		Cx486DLC		UNITS	NOTES
		MIN	MAX		
V_{OL}	Output Low Voltage $I_{OL} = 5$ mA		0.45	V	
V_{OH}	Output High Voltage $I_{OH} = -1$ mA $I_{OH} = -0.2$ mA	 2.4 V_{CC} - 0.5		V	
I_{LI}	Input Leakage Current For all pins except those listed in Table 4-1.		±15	µA	$0 < V_{IN} < V_{CC}$
I_{IH}	Input Leakage Current PEREQ		200	µA	$V_{IN} = 2.4$ Note 1
I_{IL}	Input Leakage Current A20M#, BS16#, BUSY#, ERROR#, FLUSH#, KEN#, SUSP#		- 400	µA	$V_{IL} = 0.45V$ Note 2
I_{CC}	Active I_{CC} 25 MHz (CLK2 = 50 MHz) 33 MHz (CLK2 = 66 MHz) 40 MHz (CLK2 = 80 MHz)	Typical: 435 520 560	 500 650 700	 mA mA mA	
I_{CCSM}	Suspend Mode I_{CC} 25 MHz (CLK2 = 50 MHz) 33 MHz (CLK2 = 66 MHz) 40 MHz (CLK2 = 80 MHz)	Typical: 5.0 7.5 10.0	 10.0 15.0 20.0	 mA mA mA	Note 3
I_{CCSS}	Standby I_{CC} 0 MHz (Suspended/CLK2 Stopped)	Typical: 100	 250	 µA	 Note 3
C_{IN}	Input Capacitance		10	pF	$f_C = 1$ MHz (Note 4)
C_{OUT}	Output or I/O Capacitance		12	pF	$f_C = 1$ MHz (Note 4)
C_{CLK}	CLK2 Capacitance		20	pF	$f_C = 1$ MHz (Note 4)

Notes: 1. PEREQ input has an internal pull-down resistor.
2. A20M#, BS16#, BUSY#, ERROR#, FLUSH#, KEN#, and SUSP# inputs each have an internal pull-up resistor.
3. 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).
4. Not 100% tested.

4.5 AC Characteristics

Tables 4-7, 4-8 and 4-9 list the AC characteristics including output delays, input setup requirements, input hold requirements and output float delays. These measurements are based on the measurement points identified in Figures 4-1 and 4-2. The rising clock edge reference level V_{REFC}, and other reference levels are shown in Table 4-6 below. Input or output signals must cross these levels during testing.

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

The outputs: A31-A2, ADS#, BE3#-BE0#, D/C#, HLDA, LOCK#, M/IO#, RPLVAL#, and W/R# change only at the beginning of phase one (Figure 4-1). D31-D0 (write cycles), RPLSET and SUSPA# change at the beginning of phase two.

The inputs: BUSY#, D31-D0 (read cycles), ERROR#, HOLD, PEREQ, and READY# are sampled at the beginning of phase one (Figure 4-1). A20M#, BS16#, FLUSH#, INTR, KEN#, NA#, NMI, and SUSP# are sampled at the beginning of phase two.

Table 4-6. Measurement Points for Switching Characteristics

SYMBOL	Cx486DLC	UNITS
V_{REFC}	2	V
V_{REF}	1.5	V
V_{IHC}	V_{CC} - 0.8	V
V_{ILC}	0.8	V
V_{IHD}	3	V
V_{ILD}	0	V

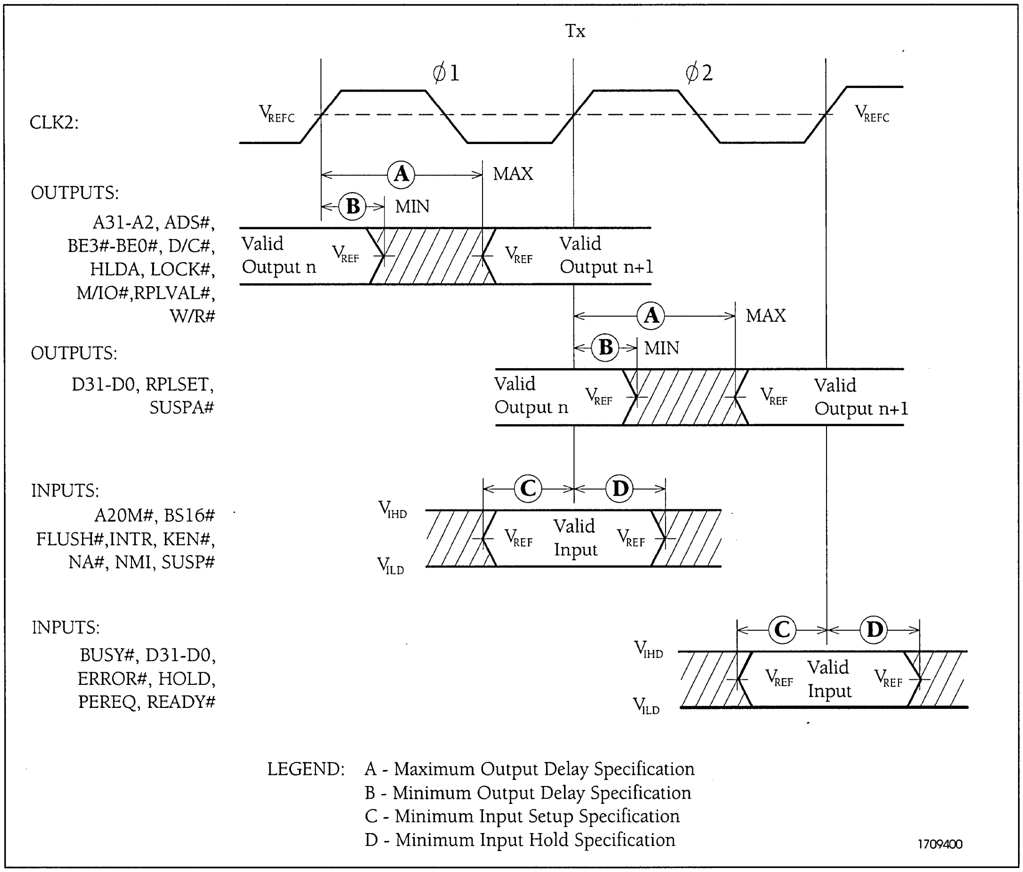

Tx
ϕ1
ϕ2
CLK2:
V_{REFC}
V_{REFC}
OUTPUTS:
A31-A2, ADS#, BE3#-BE0#, D/C#, HLDA, LOCK#, M/IO#,RPLVAL#, W/R#
A
MAX
B
MIN
Valid Output n
V_{REF}
V_{REF}
Valid Output n+1
OUTPUTS:
D31-D0, RPLSET, SUSPA#
A
MAX
B
MIN
Valid Output n
V_{REF}
V_{REF}
Valid Output n+1
INPUTS:
A20M#, BS16# FLUSH#,INTR, KEN#, NA#, NMI, SUSP#
C
D
V_{IHD}
V_{ILD}
V_{REF}
Valid Input
V_{REF}
INPUTS:
BUSY#, D31-D0, ERROR#, HOLD, PEREQ, READY#
C
D
V_{IHD}
V_{ILD}
V_{REF}
Valid Input
V_{REF}
LEGEND: A - Maximum Output Delay Specification
B - Minimum Output Delay Specification
C - Minimum Input Setup Specification
D - Minimum Input Hold Specification
1709400

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

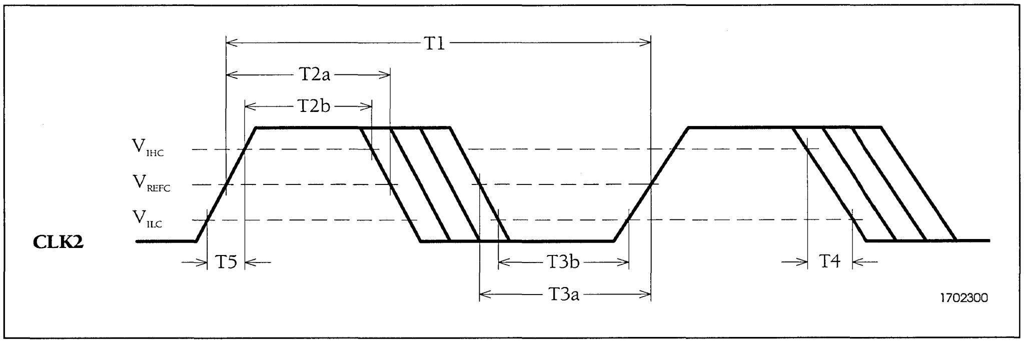


Figure 4-2. CLK2 Timing Measurement Points

Table 4-7. AC Characteristics for Cx486DLC-25

$V_{CC} = 5.0\,V \pm 5\%$, $T_C = 0° \text{ to } 85°C$

SYMBOL	PARAMETER	25 MHz		FIGURE	NOTES
		MIN (ns)	MAX (ns)		
T1	CLK2 Period	20		4-2	Note 1
T2a	CLK2 High Time	7		4-2	Note 2
T2b	CLK2 High Time	4		4-2	Note 2
T3a	CLK2 Low Time	7		4-2	Note 2
T3b	CLK2 Low Time	5		4-2	Note 2
T4	CLK2 Fall Time		7	4-2	Note 2
T5	CLK2 Rise Time		7	4-2	Note 2
T6	A31- A2 Valid Delay	4	21	4-4, 4-7	C_L = 50 pF
T7	A31- A2 Float Delay	4	30	4-6	Note 3
T8	BE3# - BE0#, LOCK# Valid Delay	4	21	4-4, 4-7	C_L = 50 pF
T9	BE3# - BE0#, LOCK# Float Delay	4	30	4-6	Note 3
T10	ADS#, D/C#, M/IO#, RPLVAL#, W/R# Valid Delay	4	21	4-4, 4-7	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, RPLVAL#, W/R# Float Delay	4	30	4-7	Note 3
T12	D31-D0 Write Data, RPLSET, SUSPA# Valid Delay	7	27	4-4, 4-5	C_L = 50 pF, Note 5
T12a	D31-D0 Write Data Hold Time	2		4-6	
T13	D31-D0 Write Data, RPLSET Float Delay	4	22	4-7	Note 3
T14	HLDA Valid Delay	4	22	4-7	C_L = 50 pF
T15	A20M#, FLUSH#, KEN#, NA#, SUSP# Setup Time	5		4-3	Note 4
T16	A20M#, FLUSH#, KEN#, NA#, SUSP# Hold Time	3		4-3	Note 4
T17	BS16# Setup Time	7		4-3	
T18	BS16# Hold Time	3		4-3	
T19	READY# Setup Time	9		4-3	
T20	READY# Hold Time	4		4-3	
T21	D31-D0 Read Data Setup Time	7		4-3	
T22	D31-D0 Read Data Hold Time	5		4-3	
T23	HOLD Setup Time	9		4-3	
T24	HOLD Hold Time	3		4-3	
T25	RESET Setup Time	8		4-8	
T26	RESET Hold Time	3		4-8	
T27	NMI, INTR Setup Time	6		4-3	Note 4
T28	NMI, INTR Hold Time	6		4-3	Note 4
T29	PEREQ, ERROR#, BUSY# Setup Time	6		4-3	Note 4
T30	PEREQ, ERROR#, BUSY# Hold Time	5		4-3	Note 4

Notes are located on page 4-11

Table 4-8. AC Characteristics for Cx486DLC-33

V_{CC} = 5.0 V ± 5%, T_C = 0° to 85°C

SYMBOL	PARAMETER	33 MHz		FIGURE	NOTES
		MIN (ns)	MAX (ns)		
T1	CLK2 Period	15		4-2	Note 1
T2a	CLK2 High Time	6.25		4-2	Note 2
T2b	CLK2 High Time	4.5		4-2	Note 2
T3a	CLK2 Low Time	6.25		4-2	Note 2
T3b	CLK2 Low Time	4.5		4-2	Note 2
T4	CLK2 Fall Time		4	4-2	Note 2
T5	CLK2 Rise Time		4	4-2	Note 2
T6	A31- A2 Valid Delay	4	15	4-4, 4-7	C_L = 50 pF
T7	A31- A2 Float Delay	4	20	4-6	Note 3
T8	BE3# - BE0#, LOCK# Valid Delay	4	15	4-4, 4-7	C_L = 50 pF
T9	BE3# - BE0#, LOCK# Float Delay	4	20	4-6	Note 3
T10	ADS#, D/C#, M/IO#, RPLVAL#, W/R# Valid Delay	4	15	4-4, 4-7	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, RPLVAL#, W/R# Float Delay	4	20	4-7	Note 3
T12	D31-D0 Write Data, RPLSET, SUSPA# Valid Delay	7	24	4-4, 4-5	C_L = 50 pF, Note 5
T12a	D31-D0 Write Data Hold Time	2		4-6	
T13	D31-D0 Write Data, RPLSET Float Delay	4	17	4-7	Note 3
T14	HLDA Valid Delay	4	20	4-7	C_L = 50 pF
T15	A20M#, FLUSH#, KEN#, NA#, SUSP# Setup Time	5		4-3	Note 4
T16	A20M#, FLUSH#, KEN#, NA#, SUSP# Hold Time	2		4-3	Note 4
T17	BS16# Setup Time	5		4-3	
T18	BS16# Hold Time	2		4-3	
T19	READY# Setup Time	7		4-3	
T20	READY# Hold Time	4		4-3	
T21	D31-D0 Read Data Setup Time	5		4-3	
T22	D31-D0 Read Data Hold Time	3		4-3	
T23	HOLD Setup Time	7		4-3	
T24	HOLD Hold Time	2		4-3	
T25	RESET Setup Time	5		4-8	
T26	RESET Hold Time	2		4-8	
T27	NMI, INTR Setup Time	5		4-3	Note 4
T28	NMI, INTR Hold Time	5		4-3	Note 4
T29	PEREQ, ERROR#, BUSY# Setup Time	5		4-3	Note 4
T30	PEREQ, ERROR#, BUSY# Hold Time	4		4-3	Note 4

Notes are located on page 4-11

Table 4-9. AC Characteristics for Cx486DLC-40

$V_{CC} = 5.0\,V \pm 5\%$, $T_C = 0° \text{ to } 85°C$

SYMBOL	PARAMETER	40 MHz MIN (ns)	40 MHz MAX (ns)	FIGURE	NOTES
T1	CLK2 Period	12.5		4-2	Note 1
T2a	CLK2 High Time	5		4-2	Note 2
T2b	CLK2 High Time	3.25		4-2	Note 2
T3a	CLK2 Low Time	5		4-2	Note 2
T3b	CLK2 Low Time	3.25		4-2	Note 2
T4	CLK2 Fall Time		4	4-2	Note 2
T5	CLK2 Rise Time		4	4-2	Note 2
T6	A31- A2 Valid Delay	3	12.5	4-4, 4-7	C_L = 50 pF
T7	A31- A2 Float Delay	3	17	4-7	Note 3
T8	BE3# - BE0#, LOCK# Valid Delay	3	12.5	4-4, 4-7	C_L = 50 pF
T9	BE3# - BE0#, LOCK# Float Delay	3	17	4-7	Note 3
T10	ADS#, D/C#, M/IO#, RPLVAL#, W/R# Valid Delay	3	12.5	4-4, 4-7	C_L = 50 pF
T11	ADS#, D/C#, M/IO#, RPLVAL#, W/R# Float Delay	3	17	4-7	Note 3
T12	D31-D0 Write Data, RPLSET, SUSPA# Valid Delay	5	20	4-4, 4-5	C_L = 50 pF, Note 5
T12a	D31-D0 Write Data Hold Time	2		4-6	
T13	D31-D0 Write Data, RPLSET Float Delay	3	14.5	4-7	Note 3
T14	HLDA Valid Delay	3	17	4-7	C_L = 50 pF
T15	A20M#, FLUSH#, KEN#, NA#, SUSP# Setup Time	5		4-3	Note 4
T16	A20M#, FLUSH#, KEN#, NA#, SUSP# Hold Time	2		4-3	Note 4
T17	BS16# Setup Time	5		4-3	
T18	BS16# Hold Time	2		4-3	
T19	READY# Setup Time	5		4-3	
T20	READY# Hold Time	3		4-3	
T21	D31-D0 Read Data Setup Time	5		4-3	
T22	D31-D0 Read Data Hold Time	3		4-3	
T23	HOLD Setup Time	4		4-3	
T24	HOLD Hold Time	2		4-3	
T25	RESET Setup Time	4.5		4-8	
T26	RESET Hold Time	2		4-8	
T27	NMI, INTR Setup Time	5		4-3	Note 4
T28	NMI, INTR Hold Time	5		4-3	Note 4
T29	PEREQ, ERROR#, BUSY# Setup Time	5		4-3	Note 4
T30	PEREQ, ERROR#, BUSY# Hold Time	3		4-3	Note 4

Notes are located on page 4-11

Notes for Tables 4-7, 4-8 and 4-9:

1. Input clock can be stopped, therefore minimum CLK2 frequency is 0 MHz.
2. These parameters are not tested. They are guaranteed by design characterization.
3. Float condition occurs when maximum output current becomes less than I_{LI} in magnitude. Float is not 100% tested.
4. The following inputs are allowed to be asynchronous to CLK2: A20M#, BUSY#, ERROR#, FLUSH#, INTR, NMI and PEREQ. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
5. T12 minimum time is not 100% tested.

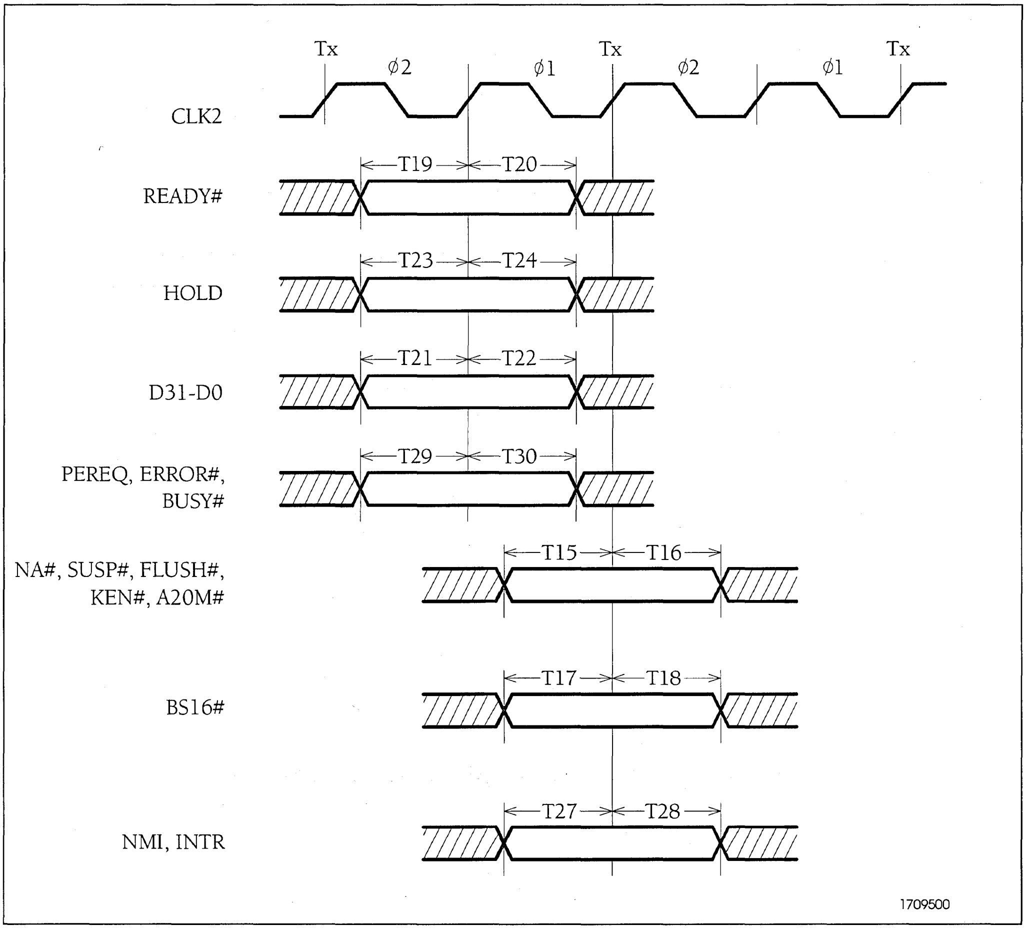


Figure 4-3. Input Signal Setup and Hold Timing

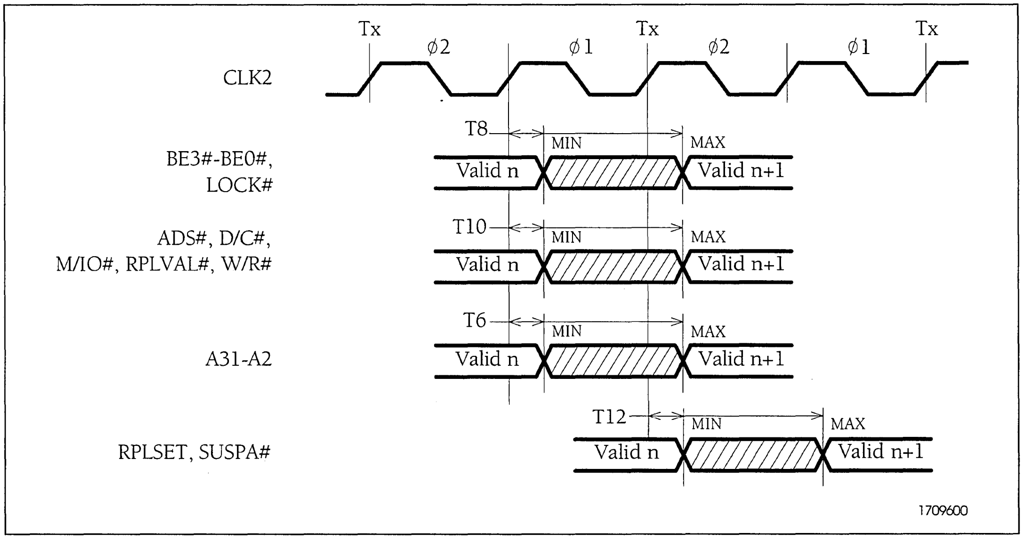


Figure 4-4. Output Signal Valid Delay Timing

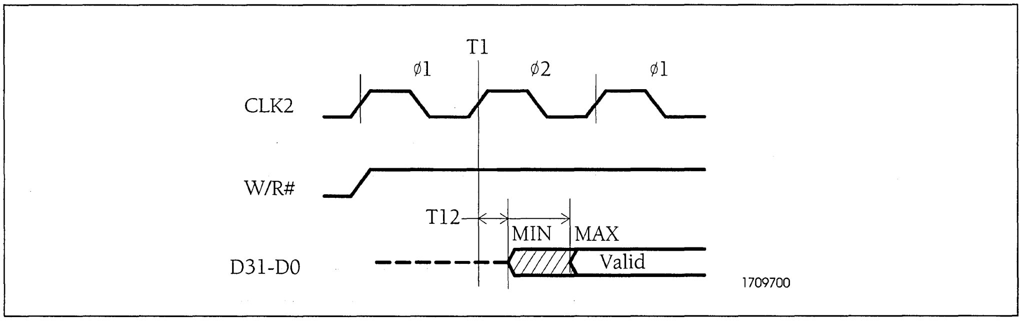


Figure 4-5. Data Write Cycle Valid Delay Timing

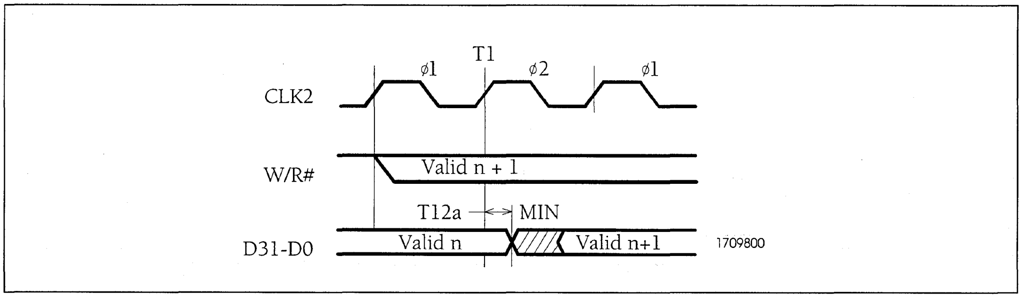


Figure 4-6. Data Write Cycle Hold Timing

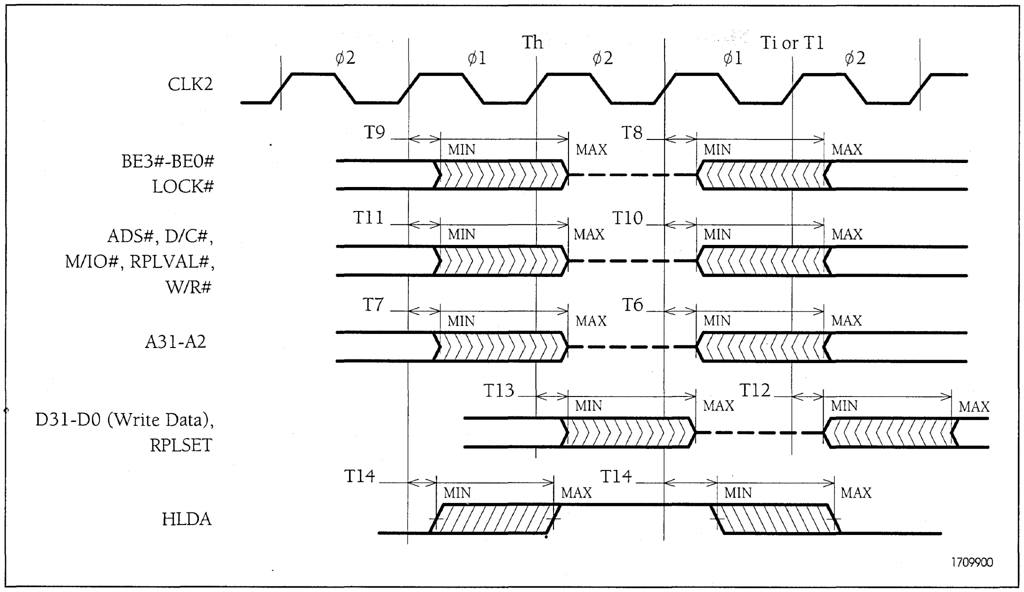


Figure 4-7. Output Signal Float Delay and HLDA Valid Delay Timing

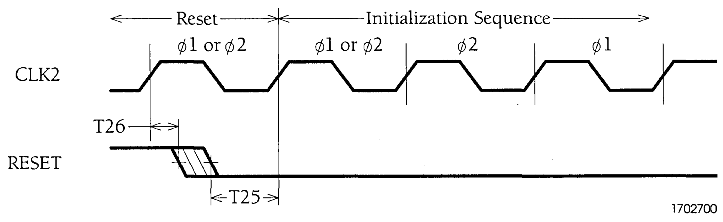


Figure 4-8. RESET Setup and Hold Timing

5. MECHANICAL SPECIFICATIONS

5.1 Pin Assignments

The pin assignments for the Cx486DLC are shown as viewed from the pin side in Figure 5-1 and as viewed from the top side (component side when mounted on a P.C. board) in Figure 5-2. The signal names are listed in Tables 5-1 and 5-2, sorted by pin number and signal name respectively.

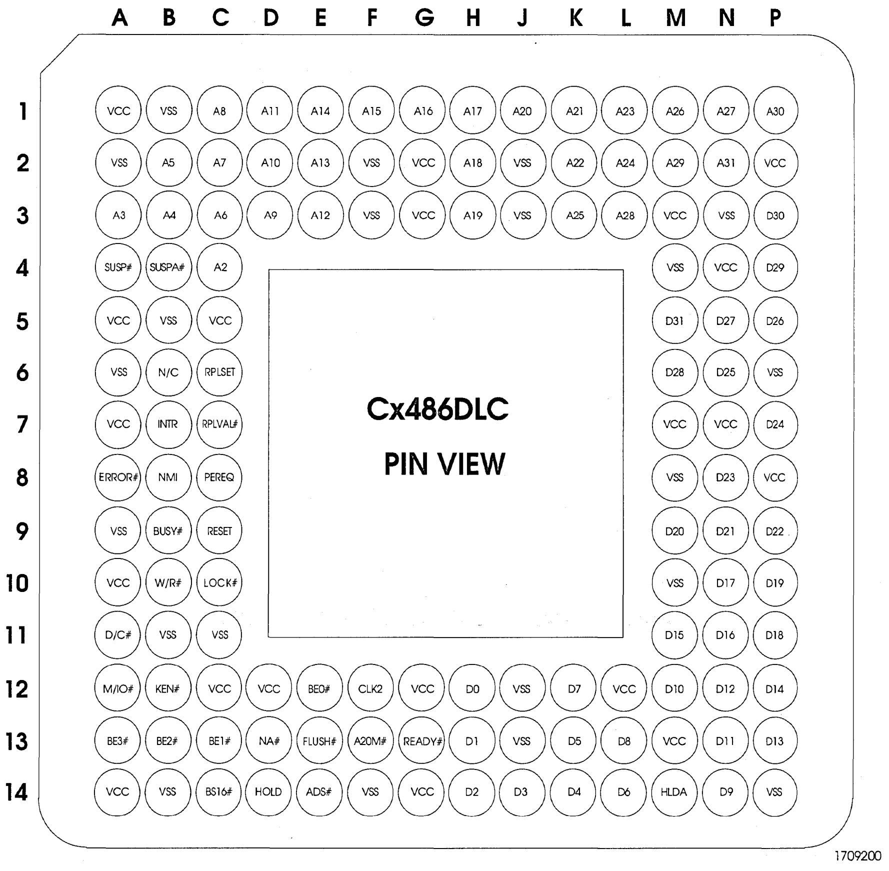

A B C D E F G H J K L M N P
1 VCC VSS A8 A11 A14 A15 A16 A17 A20 A21 A23 A26 A27 A30
2 VSS A5 A7 A10 A13 VSS VCC A18 VSS A22 A24 A29 A31 VCC
3 A3 A4 A6 A9 A12 VSS VCC A19 VSS A25 A28 VCC VSS D30
4 SUSP# SUSPA# A2 VSS VCC D29
5 VCC VSS VCC D31 D27 D26
6 VSS N/C RPLSET D28 D25 VSS
7 VCC INTR RPLVAL# VCC VCC D24
8 ERROR# NMI PEREQ VSS D23 VCC
9 VSS BUSY# RESET D20 D21 D22
10 VCC W/R# LOCK# VSS D17 D19
11 D/C# VSS VSS D15 D16 D18
12 M/IO# KEN# VCC VCC BE0# CLK2 VCC D0 VSS D7 VCC D10 D12 D14
13 BE3# BE2# BE1# NA# FLUSH# A20M# READY# D1 VSS D5 D8 VCC D11 D13
14 VCC VSS BS16# HOLD ADS# VSS VCC D2 D3 D4 D6 HLDA D9 VSS
Cx486DLC
PIN VIEW
1709200

Figure 5-1. Bottom View of Package Pins

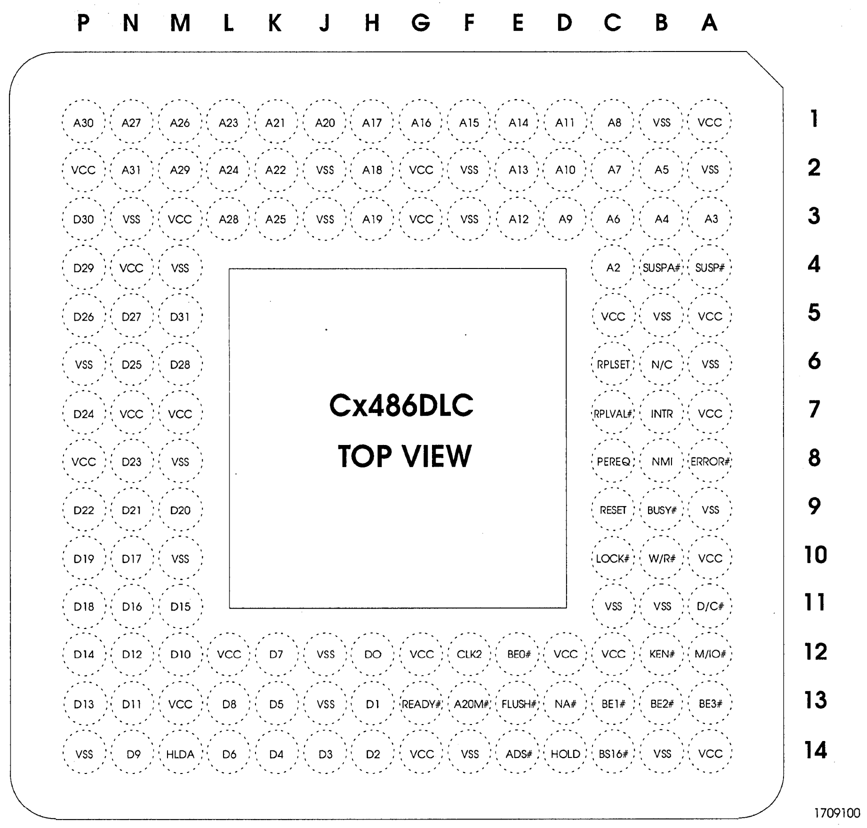

P N M L K J H G F E D C B A
1 2 3 4 5 6 7 8 9 10 11 12 13 14
A30 A27 A26 A23 A21 A20 A17 A16 A15 A14 A11 A8 VSS VCC
VCC A31 A29 A24 A22 VSS A18 VCC VSS A13 A10 A7 A5 VSS
D30 VSS VCC A28 A25 VSS A19 VCC VSS A12 A9 A6 A4 A3
D29 VCC VSS A2 SUSPA# SUSP#
D26 D27 D31 VCC VSS VCC
VSS D25 D28 RPLSET N/C VSS
D24 VCC VCC RPLVAL# INTR VCC
VCC D23 VSS PEREQ NMI ERROR#
D22 D21 D20 RESET BUSY# VSS
D19 D17 VSS LOCK# W/R# VCC
D18 D16 D15 VSS VSS D/C#
D14 D12 D10 VCC D7 VSS DO VCC CLK2 BE0# VCC VCC KEN# M/IO#
D13 D11 VCC D8 D5 VSS D1 READY# A20M# FLUSH# NA# BE1# BE2# BE3#
VSS D9 HLDA D6 D4 D3 D2 VCC VSS ADS# HOLD BS16# VSS VCC
Cx486DLC
TOP VIEW
1709100

Figure 5-2. Top View of Package Pins

PRELIMINARY

Table 5-1. Signal Names Sorted by Pin Number

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

Table 5-2. Pin Numbers Sorted by Signal Name

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

5.2 Package Dimensions

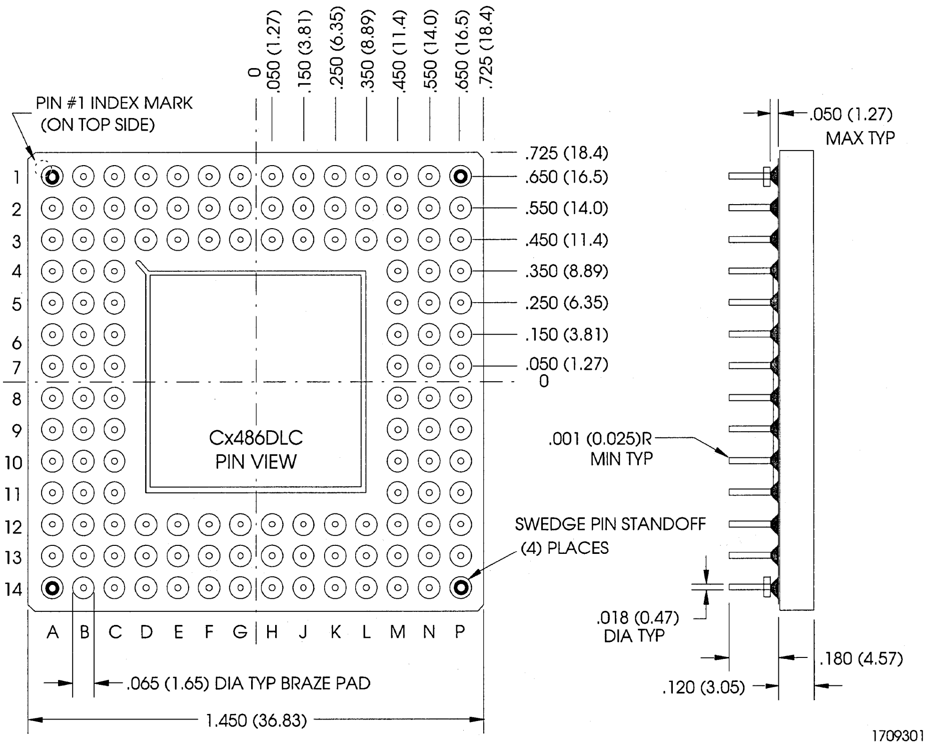


Figure 5-3. 132-Pin PGA Package Dimensions

5.3 Thermal Characteristics

The Cx486DLC is designed to operate when case temperature is between 0° - 85° C. The case temperature is measured on the top center of the package. The maximum die temperature ($T_{j\,max}$) and the maximum ambient temperature ($T_{a\,max}$) can be calculated using the following equations.

$$T_{j\,max} = T_c + (P_{max} \times \theta_{jc})$$
$$T_{a\,max} = T_j - (P_{max} \times \theta_{ja})$$

where:

$T_{j\,max}$	= Maximum average junction temperature (°C)
T_c	= Case temperature at top center of package (°C)
P_{max}	= Maximum device power dissipation (W)
θ_{jc}	= Junction-to-case thermal resistance (°C/W)
$T_{a\,max}$	= Maximum ambient temperature (°C)
T_j	= Average junction temperature (°C)
θ_{ja}	= Junction-to-ambient thermal resistance (°C/W)

Values for θ_{ja} and θ_{jc} are given in Table 5-3 for various airflows for the 132-lead ceramic pin grid array package.

Table 5-3. Package Thermal Resistance and AirFlow 132-Pin Ceramic PGA Package

AIRFLOW (FT/MIN)	THERMAL RESISTANCE (°C/W)	
	θ_{ja}	θ_{jc}
0	20	3
100	18	3
250	14	3
500	10	3

6. INSTRUCTION SET

This section summarizes the Cx486DLC instruction set and provides detailed information on the instruction encodings. All instructions are listed in the Instruction Set Summary Table (Table 6-16), which provides information on the instruction encoding, which flags are effected, and the instruction clock counts for each instruction. The clock count values are based on the assumptions described in section 6.3.

6.1 General Instruction Format

All of the Cx486DLC machine instructions follow the general instruction format shown in Figure 6-1. These instructions vary in length and can start at any byte address. An instruction consists of one or more bytes that can include: prefix byte(s), at least one opcode byte(s), mod r/m byte, s-i-b byte, address displacement byte(s) and immediate data byte(s). An instruction can be as short as one byte and as long as 15 bytes. If there are more than 15 bytes in the instruction a general protection fault (error code of 0) is generated.

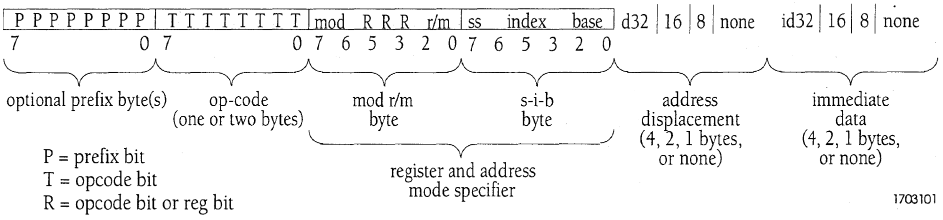


Figure 6-1. General Instruction Format

6.2 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 sign extension. All the fields are described in Table 6-1 and the subsequent paragraphs provide greater detail.

Table 6-1. Instruction Fields

FIELD NAME	DESCRIPTION	NUMBER OF BITS
Prefix	Specifies segment register override, address and operand size, repeat elements in string instruction, LOCK# assertion.	1 byte / prefix
Opcode	Identifies instruction operation.	1 or 2 bytes
w	Specifies if data is byte or full size (full size is either 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

6.2.1 Prefixes

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

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

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

Table 6-2. Instruction Prefix Summary

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

6.2.2 Opcode Field

The opcode field is either one or two bytes in length 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 3 bits in the following byte, called the MOD R/M byte, to resolve the operation type. Opcodes for the entire Cx486DLC instruction set are listed in the Instruction Set Summary Table. The opcodes are given in hex values unless shown within brackets ([ ]). Values shown in brackets are binary values.

6.2.3 w Field

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

Table 6-3. w Field Encoding

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

6.2.4 d Field

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

Table 6-4. d Field Encoding

d FIELD	DIRECTION OF OPERATION	SOURCE OPERAND	DESTINATION 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

6.2.5 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.

Table 6-5. reg Field Encoding

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

6.2.6 mod and r/m Fields

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

Table 6-6. 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	SS:[BP+SI]	DS:[EDX]
00 011	SS:[BP+DI]	DS:[EBX]
00 100	DS:[SI]	s-i-b is present (See 6.2.7)
00 101	DS:[DI]	DS:[d32]
00 110	DS:[d16]	DS:[ESI]
00 111	DS:[BX]	DS:[EDI]
01 000	DS:[BX+SI+d8]	DS:[EAX+d8]
01 001	DS:[BX+DI+d8]	DS:[ECX+d8]
01 010	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 6.2.7)
01 101	DS:[DI+d8]	SS:[EBP+d8]
01 110	SS:[BP+d8]	DS:[ESI+d8]
01 111	DS:[BX+d8]	DS:[EDI+d8]
10 000	DS:[BX+SI+d16]	DS:[EAX+d32]
10 001	DS:[BX+DI+d16]	DS:[ECX+d32]
10 010	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 6.2.7)
10 101	DS:[DI+d16]	SS:[EBP+d32]
10 110	SS:[BP+d16]	DS:[ESI+d32]
10 111	DS:[BX+d16]	DS:[EDI+d32]
11 000-11 111	See Table 6-6A	See Table 6-6A

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

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

6.2.7 mod and base Fields

In Table 6-6A, the note "s-i-b present" for certain entries forces the use of the mod and base field as listed in Table 6-7.

Table 6-7. mod base Field Encoding

mod base	32-BIT ADDRESS MODE with mod r/m and s-i-b Bytes Present
00 000	DS:[EAX+(scaled index)]
00 001	DS:[ECX+(scaled index)]
00 010	DS:[EDX+(scaled index)]
00 011	DS:[EBX+(scaled index)]
00 100	SS:[ESP+(scaled index)]
00 101	DS:[d32+(scaled index)]
00 110	DS:[ESI+(scaled index)]
00 111	DS:[EDI+(scaled index)]
01 000	DS:[EAX+(scaled index)+d8]
01 001	DS:[ECX+(scaled index)+d8]
01 010	DS:[EDX+(scaled index)+d8]
01 011	DS:[EBX+(scaled index)+d8]
01 100	SS:[ESP+(scaled index)+d8]
01 101	SS:[EBP+(scaled index)+d8]
01 110	DS:[ESI+(scaled index)+d8]
01 111	DS:[EDI+(scaled index)+d8]
10 000	DS:[EAX+(scaled index)+d32]
10 001	DS:[ECX+(scaled index)+d32]
10 010	DS:[EDX+(scaled index)+d32]
10 011	DS:[EBX+(scaled index)+d32]
10 100	SS:[ESP+(scaled index)+d32]
10 101	SS:[EBP+(scaled index)+d32]
10 110	DS:[ESI+(scaled index)+d32]
10 111	DS:[EDI+(scaled index)+d32]

6.2.8 ss Field

The ss field (Table 6-8) specifies the scale factor used in the offset mechanism for address calculation. The scale factor multiplies the index value to provide one of the components used to calculate the offset address.

Table 6-8. ss Field Encoding

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

6.2.9 index Field

The index field (Table 6-9) specifies the index register used by the offset mechanism for offset address calculation. When no index register is used (index field = 100), the ss value must be 00 or the effective address is undefined.

Table 6-9. index Field Encoding

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

6.2.10 sreg2 Field

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

Table 6-10. sreg2 Field Encoding

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

6.2.11 sreg3 Field

The sreg3 field (Table 6-11) is 3-bit field that is similar to the sreg2 field, but allows use of the FS and GS segment registers.

Table 6-11. sreg3 Field Encoding

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

6.2.12 eee Field

The eee field is used to select the control, debug and test registers as indicated in Table 6-12. The values shown in Table 6-12 are the only valid encodings for the eee bits.

Table 6-12. eee Field Encoding

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

6.3 Flags

The Instruction Set Summary Table lists nine flags that are affected by the execution of instuctions. The conventions shown in Table 6-13 are used to identify the different flags. Table 6-14 lists the conventions used to indicate what action the instuction has on the particular flag.

Table 6-13. Flag Abbreviations

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

Table 6-14. Action of Instruction on Flag

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

6.4 Clock Counts

6.4.1 Assumptions

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

1. The instruction has been prefetched, decoded and is ready for execution.
2. Bus cycles do not require wait states.
3. There are no local bus HOLD requests delaying processor access to the bus.
4. No exceptions are detected during instruction execution.
5. 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 clock to the clock count shown.
6. All clock counts assume aligned 32-bit memory/IO operands for cache miss counts.
7. If instructions access a misaligned 32-bit operand address, add 2 clocks for read or write and add 4 clock counts for read and write.

6.4.2 Abbreviations

The clock counts 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 6-15.

Table 6-15. Clock Count Abbreviations

CLOCK COUNT SYMBOL	EXPLANATION
/	Register operand/memory operand.
n	Number of times operation is repeated.
L	Level of the stack frame.
|	Condition jump taken | conditional jump not taken.
\	CPL ≤ IOPL \ CPL > IOPL.

Table 6-16. Instruction Set Summary

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
AAA *ASCII Adjust AL after Add*	37	- - - - - - x - x	4		4			
AAD *ASCII Adjust AX before Divide*	D5 0A	- - - - x x - x -	4		4			
AAM *ASCII Adjust AX after Multiply*	D4 0A	- - - - x x - x -	16		16			
AAS *ASCII Adjust AL after Subtract*	3F	- - - - - - x - x	4		4			
ADC *Add with Carry*		x - - - x x x x x					b	h
Register to Register	1 [00dw] [11 reg r/m]		1		1			
Register to Memory	1 [000w] [mod reg r/m]		3	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*		x - - - x x x x x					b	h
Register to Register	0 [00dw] [11 reg r/m]		1		1			
Register to Memory	0 [000w] [mod reg r/m]		3	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 - - - x x - x 0					b	h
Register to Register	2 [00dw] [11 reg r/m]		1		1			
Register to Memory	2 [000w] [mod reg r/m]		3	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*		- - - - - x - - -					a	h
From Register/Memory	63 [mod reg r/m]				6/10	10		

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

Cyrix® Advancing the Standards

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
BOUND *Check Array Boundaries* If Out of Range (Int 5) If In Range	62 [mod reg r/m]	- - - - - - - - - -	 11+int 11		 11+int 11		b,e	g,h,j,k,r
BSF *Scan Bit Forward* Register/Memory, Register	 0F BC [mod reg r/m]	- - - - - x - - -	 5/7+n	 9+n	 5/7+n	 9+n	b	h
BSR *Scan Bit Reverse* Register/Memory, Register	 0F BC [mod reg r/m]	- - - - - x - - -	 5/7+n	 9+n	 5/7+n	 9+n	b	h
BSWAP *Byte Swap*	0F C [1 reg]	- - - - - - - - - -	4		4			
BT *Test Bit* Register/Memory, Immediate Register/Memory, Register	 0F BA [mod 100 r/m]# 0F A3 [mod reg r/m]	- - - - - - - - x	 3/4 3/6	 5 7	 3/4 3/6	 5 7	b	h
BTC *Test Bit and Complement* Register/Memory, Immediate Register/Memory, Register	 0F BA [mod 111 r/ 0F m]# BB [mod reg r/m]	- - - - - - - - x	 4/5 5/8	 6 9	 4/5 5/8	 6 9	b	h
BTR *Test Bit and Reset* Register/Memory, Immediate Register/Memory, Register	 0F BA [mod 110 r/m]# 0F B3 [mod reg r/m]	- - - - - - - - x	 4/5 5/8	 6 9	 4/5 5/8	 6 9	b	h
BTS *Test Bit and Set* Register/Memory Register (short form)	 0F BA [mod 101 r/m] 0F AB [mod reg r/m]	- - - - - - - - x	 3/5 4/7	 6 8	 3/5 4/7	 6 8	b	h

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		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*		- - - - - - - - -					b	h,j,k,r
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,		12		30			
Call Gate to Same Privilege	selector]				41	41		
Call Gate to Different Privilege No P					83	83		
Call Gate to Different Privilege x P's					81+4x	81+4x		
16-bit Task to 16-bit TSS					235	235		
16-bit Task to 32-bit TSS					262	265		
16-bit Task to V86 Task					179	182		
32-bit Task to 16-bit TSS					238	238		
32-bit Task to 32-bit TSS					265	268		
32-bit Task to V86 Task					182	185		
Indirect Intersegment	FF [mod 011 r/m]		14	17	14	34		
Call Gate to Same Privilege					43	43		
Call Gate to Different Privilege No P					85	85		
Call Gate to Different Privilege Level x P's					86+4x	86+4x		
16-bit Task to 16-bit TSS					237	240		
16-bit Task to 32-bit TSS					264	270		
16-bit Task to V86 Task					181	187		
32-bit Task to 16-bit TSS					240	243		
32-bit Task to 32-bit TSS					267	273		
32-bit Task to V86 Task					184	190		
CBW *Convert Byte to Word*	98	- - - - - - - - -	3		3			
CDQ *Convert Doubleword to Quadword*	99	- - - - - - - - -	1		1			
CLC *Clear Carry Flag*	F8	- - - - - - - - 0	1		1			
CLD *Clear Direction Flag*	FC	- 0 - - - - - - -	1		1			
CLI *Clear Interrupt Flag*	FA	- - 0 - - - - - -	7		7			m
CLTS *Clear Task Switched Flag*	0F 06	- - - - - - - - -	5		5		c	l

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
CMC *Complement the Carry Flag*	F5	- - - - - - - - x	1		1			
CMP *Compare Integers*		x - - - x x x x x					b	h
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			
CMPS *Compare String*	A [011w]	x - - - x x x x x	7	8	7	8	b	h
CMPXCHG *Compare and Exchange*		x - - - x x x x x						
Register1, Register2	0F B [000w] [11 reg2 reg1]		5		5			
Memory, Register	0F B [000w] [mod reg r/m]		7	8	7	8		
CWD *Convert Word to Doubleword*	99	- - - - - - - - -	1		1			
CWDE *Convert Word to Doubleword Extended*	98	- - - - - - - - -	3		3			
DAA *Decimal Adjust AL after Add*	27	- - - - x x x x x	4		4			
DAS *Decimal Adjust AL after Subtract*	2F	- - - - x x x x x	4		4			
DEC *Decrement by 1*		x - - - x x x x -					b	h
Register/Memory	F [111w] [mod 001 r/m]		1/3	5	1/3	5		
Register (short form)	4 [1 reg]		1		1			
DIV *Unsigned Divide*	F [011w] [mod 110 r/m]	- - - - - - - - -					b,e	e,h
Accumulator by Register/Memory								
Divisor: Byte			14/15	17	14/15	17		
Word			22/23	24	22/23	24		
Doubleword			38/39	40	38/39	40		

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
ENTER *Enter New Stack Frame*	C8 ++ [8-bit Level]	- - - - - - - - -					b	h
Level = 0			7		7			
Level = 1			10	10	10	10		
Level (L) > 1			6+4*L	6+4*L	6+4*L	6+4*L		
HLT *Halt*	F4	- - - - - - - - -	3		3			l
IDIV *Integer (Signed) Divide*		- - - - - - - - -					b,e	e,h
Accumulator by Register/Memory	F [011w] [mod 111 r/m]							
Divisor: Byte			19/20	22	19/20	22		
Word			27/28	29	27/28	29		
Doubleword			43/44	47	43/44	47		
IMUL *Integer (Signed) Multiply*		x - - - - - - - x					b	h
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*		- - - - - - - - -						m
Fixed Port	E [010w] [port number]		16	16	6\19	6\20		
Variable Port	E [110w]		16	16	6\19	6\20		
INC *Increment by 1*		x - - - x x x x -					b	h
Register/Memory	F [111w] [mod 000 r/m]		1/3	5	1/3	5		
Register (short form)	4 [0 reg]		1		1			

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
INS *Input String from I/O Port*	6 [110w]	- - - - - - - - -	20	20	6/19	6/20	b	h, m
INT *Software Interrupt*		- x 0 - - - - - -					b,e	g,j,k,r
INT i	CD [i]		14	16				
Protected Mode:								
Interrupt or Trap to Same Privilege					49	50		
Interrupt or Trap to Different Privilege					77	78		
16-bit Task to 16-bit TSS by Task Gate					233	234		
16-bit Task to 32-bit TSS by Task Gate					260	264		
16-bit Task to V86 by Task Gate					177	181		
16-bit Task to 16-bit TSS by Task Gate					236	237		
32-bit Task to 32-bit TSS by Task Gate					263	267		
32-bit Task to V86 by Task Gate					180	184		
V86 to 16-bit TSS by Task Gate					236	237		
86 to 32-bit TSS by Task Gate					263	267		
86 to Privilege 0 by Trap Gate/Int Gate					93	94		
INT 3	CC		14	16				
Protected Mode:								
Interrupt or Trap to Same Privilege					49	50		
Interrupt or Trap to Different Privilege					77	78		
16-bit Task to 16-bit TSS by Task Gate					233	234		
16-bit Task to 32-bit TSS by Task Gate					260	264		
16-bit Task to V86 by Task Gate					177	181		
32-bit Task to 16-bit TSS by Task Gate					236	237		
32-bit Task to 32-bit TSS by Task Gate					263	267		
32-bit Task to V86 by Task Gate					180	184		
V86 to 16-bit TSS by Task Gate					236	237		
V86 to 32-bit TSS by Task Gate					263	267		
V86 to Privilege 0 by Trap Gate/Int Gate					93	94		
Continued on the next page ...								

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		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)**		- - x 0 - - - - -					b,e	g,j,k,r
INTO	CE							
If OF==0			1	1	1	1		
If OF==1 (INT 4)			15	17				
Protected Mode:								
Interrupt or Trap to Same Privilege					49	50		
Interrupt or Trap to Different Privilege					77	78		
16-bit Task to 16-bit TSS by Task Gate					233	234		
16-bit Task to 32-bit TSS by Task Gate					260	264		
16-bit Task to V86 by Task Gate					177	181		
32-bit Task to 16-bit TSS by Task Gate					236	237		
32-bit Task to 32-bit TSS by Task Gate					263	267		
32-bit Task to V86 by Task Gate					180	184		
V86 to 16-bit TSS by Task Gate					236	237		
V86 to 32-bit TSS by Task Gate					263	267		
V86 to Privilege 0 by Trap Gate/Int Gate					93	94		
INVD *Invalidate Cache*	0F 08	- - - - - - - - - -	4		4			
INVLPG *Invalidate TLB Entry*	0F 01 [mod 111 r/m]	- - - - - - - - - -	4		4			
IRET *Interrupt Return*	CF	x x x x x x x x x						g,h,j,k,r
Real Mode			14	14				
Protected Mode:								
Within Task to Same Privilege					31	33		
Within Task to Different Privilege					66	70		
16-bit Task to 16-bit Task					229	232		
16-bit Task to 32-bit TSS					256	262		
16-bit Task to V86 Task					173	179		
32-bit Task to 16-bit TSS					232	235		
32-bit Task to 32-bit TSS					259	267		
32-bit Task to V86 Task					176	182		

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
JB/JNAE/JC *Jump on Below/Not Above or Equal/ Carry*		- - - - - - - - -						r
8-bit Displacement	72 +		6 | 1		6 | 1			
Full Displacement	0F 82 +++		6 | 1		6 | 1			
JBE/JNA *Jump on Below or Equal/Not Above*		- - - - - - - - -						r
8-bit Displacement	76 +		6 | 1		6 | 1			
Full Displacement	0F 86 +++		6 | 1		6 | 1			
JCXZ *Jump on CX Zero*	E3 +	- - - - - - - - -	7 | 3		7 | 3			r
JE/JZ *Jump on Equal/Zero*		- - - - - - - - -						r
8-bit Displacement	74 +		6 | 1		6 | 1			
Full Displacement	0F 84 +++		6 | 1		6 | 1			
JECXZ *Jump on ECX Zero*	E3 +	- - - - - - - - -	7 | 3		7 | 3			r
JL/JNGE *Jump on Less/Not Greater or Equal*		- - - - - - - - -						r
8-bit Displacement	7C +		6 | 1		6 | 1			
Full Displacement	0F 8C +++		6 | 1		6 | 1			
JLE/JNG *Jump on Less or Equal/Not Greater*		- - - - - - - - -						r
8-bit Displacement	7E +		6 | 1		6 | 1			
Full Displacement	0F 8E +++		6 | 1		6 | 1			

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
JMP *Unconditional Jump*		- - - - - - - - -					b	h,j,k,r
Short	EB +		6		6			
Direct within Segment	E9 +++		6		6			
Register/Memory Indirect Within Segment	FF [mod 100 r/m]		6/8	10	6/8	10		
Direct Intersegment	EA [full offset, selector]		9		26			
Call Gate Same Privilege Level					37	37		
16-bit Task to 16-bit TSS					238	238		
16-bit Task to 32-bit TSS					265	268		
16-bit Task to V86 Task					182	185		
32-bit Task to 16-bit TSS					241	241		
32-bit Task to 32-bit TSS					268	271		
32-bit Task to V86 Task					185	188		
Indirect Intersegment	FF [mod 101 r/m]		11	14	30	30		
Call Gate Same Privilege Level					39	39		
16-bit Task to 16-bit TSS					240	243		
16-bit Task to 32-bit TSS					267	273		
16-bit Task to V86 Task					184	190		
32-bit Task to 16-bit TSS					243	246		
32-bit Task to 32-bit TSS					270	276		
32-bit Task to V86 Task					187	193		
JNB/JAE/JNC *Jump on Not Below/Above or*		- - - - - - - - -						r
Equal/Not Carry			6 | 1		6 | 1			
8-bit Displacement	73 +		6 | 1		6 | 1			
Full Displacement	0F 83 +++							
JNBE/JA *Jump on Not Below or Equal/Above*		- - - - - - - - -						r
8-bit Displacement	77 +		6 | 1		6 | 1			
Full Displacement	0F 87 +++		6 | 1		6 | 1			
JNE/JNZ *Jump on Not Equal/Not Zero*		- - - - - - - - -						r
8-bit Displacement	75 +		6 | 1		6 | 1			
Full Displacement	0F 85 +++		6 | 1		6 | 1			
JNL/JGE *Jump on Not Less/Greater or Equal*		- - - - - - - - -						r
8-bit Displacement	7D +		6 | 1		6 | 1			
Full Displacement	0F 8D +++		6 | 1		6 | 1			

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
JNLE/JG *Jump on Not Less or Equal/Greater*		- - - - - - - - -						r
8-bit Displacement	7F +		6 | 1		6 | 1			
Full Displacement	0F 8F +++		6 | 1		6 | 1			
JNO *Jump on Not Overflow*		- - - - - - - - -						r
8-bit Displacement	71 +		6 | 1		6 | 1			
Full Displacement	0F 81 +++		6 | 1		6 | 1			
JNP/JPO *Jump on Not Parity/Parity Odd*		- - - - - - - - -						r
8-bit Displacement	7B +		6 | 1		6 | 1			
Full Displacement	0F 8B +++		6 | 1		6 | 1			
JNS *Jump on Not Sign*		- - - - - - - - -						r
8-bit Displacement	79 +		6 | 1		6 | 1			
Full Displacement	0F 89 +++		6 | 1		6 | 1			
JO *Jump on Overflow*		- - - - - - - - -						r
8-bit Displacement	70 +		6 | 1		6 | 1			
Full Displacement	0F 80 +++		6 | 1		6 | 1			
JP/JPE *Jump on Parity/Parity Even*		- - - - - - - - -						r
8-bit Displacement	7A +		6 | 1		6 | 1			
Full Displacement	0F 8A +++		6 | 1		6 | 1			
JS *Jump on Sign*		- - - - - - - - -						r
8-bit Displacement	78 +		6 | 1		6 | 1			
Full Displacement	0F 88 +++		6 | 1		6 | 1			
LAHF *Load AH with Flags*	9F	- - - - - - - - -	2		2			
LAR *Load Access Rights*		- - - - - x - - -					a	g,h,j,p
From Register/Memory	0F 02 [mod reg r/m]				11/12	14		

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
LDS *Load Pointer to DS*	C5 [mod reg r/m]	- - - - - - - - -	6	7	19	20	b	h,i,j
LEA *Load Effective Address* No Index Register With Index Register	8D [mod reg r/m]	- - - - - - - - -	 2 3		 2 3			
LEAVE *Leave Current Stack Frame*	C9	- - - - - - - - -	3	4	3	4	b	h
LES *Load Pointer to ES*	C4 [mod reg r/m]	- - - - - - - - -	6	7	19	20	b	h,i,j
LFS *Load Pointer to FS*	0F B4 [mod reg r/m]	- - - - - - - - -	6	7	19	20	b	h,i,j
LGDT *Load GDT Register*	0F 01 [mod 010 r/m]	- - - - - - - - -	9	9	9	9	b,c	h,l
LGS *Load Pointer to GS*	0F B5 [mod reg r/m]	- - - - - - - - -	6	7	19	20	b	h,i,j
LIDT *Load IDT Register*	0F 01 [mod 011 r/m]	- - - - - - - - -	9	9	9	9	b,c	h,l
LLDT *Load LDT Register* From Register/Memory	 0F 00 [mod 010 r/m]	- - - - - - - - -			 16/17	 18	a	g,h,j,l
LMSW *Load Machine Status Word* From Register/Memory	 0F 01 [mod 110 r/m]	- - - - - - - - -	 5	 8	 5	 8	b,c	h,l
LODS *Load String*	A [110w]	- - - - - - - - -	4	4	4	4	b	h
LOOP *Offset Loop/No Loop*	E2 +	- - - - - - - - -	9 | 3		9 | 3			r
LOOPNZ/LOOPNE *Offset*	E0 +	- - - - - - - - -	9 | 3		9 | 3			r

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
LOOPZ/LOOPE *Offset*	E1 +	- - - - - - - - - -	9 | 3		9 | 3			r
LSL *Load Segment Limit*		- - - - - - x - - -					a	g,h,j,p
From Register/Memory	0F 03 [mod reg r/m]				14/15	17		
LSS *Load Pointer to SS*	0F B2 [mod reg r/m]	- - - - - - - - - -	6	7	19	20	a	h,i,j
LTR *Load Task Register*		- - - - - - - - - -					a	g,h,j,l
From Register/Memory	0F 00 [mod reg r/m]				16/17	18		
MOV *Move Data*		- - - - - - - - - -					b	h,i,j
Register to Register/Memory	8 [100w] [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] +++		1/2	2	1/2	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]		1/2	2	1/2	2		
MOV *Move to/from Control/Debug/Test Regs*		- - - - - - - - - -						l
Register to CR0/CR2/CR3	0F 22 [11 eee reg]		11/3/3		11/3/3			
CR0/CR2/CR3 to Register	0F 20 [11 eee reg]		1/3/3		1/3/3			
Register to DR0-DR3	0F 23 [11 eee reg]		1		1			
DR0-DR3 to Register	0F 21 [11 eee reg]		3		3			
Register to DR6-DR7	0F 23 [11 eee reg]		1		1			
DR6-DR7 to Register	0F 21 [11 eee reg]		3		3			
Register to TR3-5	0F 26 [11 eee reg]		5		5			
TR3-5 to Register	0F 24 [11 eee reg]		5		5			
Register to TR6-TR7	0F 26 [11 eee reg]		1		1			
TR6-TR7 to Register	0F 24 [11 eee reg]		3		3			
MOVS *Move String*	A [010w]	- - - - - - - - - -	5	5	5	5	b	h

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
MOVSX *Move with Sign Extension*		- - - - - - - - -					b	h
Register from Register/Memory	0F B [111w] [mod reg r/m]		1/3	5	1/3	5		
MOVZX *Move with Zero Extension*		- - - - - - - - -					b	h
Register from Register/Memory	0F B [011w] [mod reg r/m]		2/3	5	2/3	5		
MUL *Unsigned Multiply*	F [011w] [mod 100 r/m]	x - - - - - - - x					b	h
Accumulator with Register/Memory								
Multiplier - Byte			3/5	7	3/5	7		
- Word			3/5	7	3/5	7		
- Doubleword			7/9	13	7/9	13		
NEG *Negate Integer*	F [011w] [mod 011 r/m]	x - - - x x x x x	1/3	5	1/3	5	b	h
NOP *No Operation*	90	- - - - - - - - -	3		3			
NOT *Boolean Complement*	F [011w] [mod 010 r/m]	- - - - - - - - -	1/3	5	1/3	5	b	h
OR *Boolean OR*		0 - - - x x x x 0					b	h
Register to Register	0 [10dw] [11 reg r/m]		1		1			
Register to Memory	0 [100w] [mod reg r/m]		3	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*		- - - - - - - - -						m
Fixed Port	E [011w] [port number]		18	18	4\17	4\18		
Variable Port	E [111w]		18	18	4\17	4\18		
OUTS *Output String*	6 [111w]	- - - - - - - - -	20	20	6\19	6\19	b	h,m

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		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*		- - - - - - - - -					b	h,i,j
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]		4	5	18	19		
Segment Register (ES, CS, SS, DS, FS, GS)	0F [10 sreg3 001]		4	5	18	19		
POPA *Pop All General Registers*	61	- - - - - - - - -	18	18	18	18	b	h
POPF *Pop Stack into FLAGS*	9D	x x x x x x x x x	4	5	4	5	b	h,n
PREFIX BYTES		- - - - - - - - -						m
Assert Hardware LOCK Prefix	F0							
Address Size Prefix	67							
Operand Size Prefix	66							
Segment Override Prefix								
CS	2E							
DS	3E							
ES	26							
FS	64							
GS	65							
SS	36							
PUSH *Push Value onto Stack*		- - - - - - - - -					b	h
Register/Memory	FF [mod 110 r/m]		2/4	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	- - - - - - - - -	17	17	17	17	b	h
PUSHF *Push FLAGS Register*	9C	- - - - - - - - -	2	2	2	2	b	h

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
RCL *Rotate Through Carry Left*		x - - - - - - - x					b	h
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*		x - - - - - - - x					b	h
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		
REP INS *Input String*	F2 6 [110w]	- - - - - - - - -	20+9n	20+9n	5+9n \ 18+9n	5+9n \ 19+9n	b	h,m
REP LODS *Load String*	F2 A [110w]	- - - - - - - - -	4+5n	4+5n	4+5n	4+5n	b	h
REP MOVS *Move String*	F2 A [010w]	- - - - - - - - -	5+4n	5+4n	5+4n	5+4n	b	h
REP OUTS *Output String*	F2 6 [111w]	- - - - - - - - -	20+4n	20+4n	5+4n \ 18+4n	5+4n \ 19+4n	b	h,m
REP STOS *Store String*	F2 A [101w]	- - - - - - - - -	3+4n	3+4n	3+4n	3+4n	b	h
REPE CMPS *Compare String (Find non-match)*	F3 A [011w]	x - - - x x x x x	5+8n	5+8n	5+8n	5+8n	b	h
REPE SCAS *Scan String (Find non-AL/AX/EAX)*	F3 A [111w]	x - - - x x x x x	4+5n	4+6n	4+5n	4+6n	b	h
REPNE CMPS *Compare String (Find match)*	F2 A [011w]	x - - - x x x x x	5+8n	5+8n	5+8n	5+8n	b	h
REPNE SCAS *Scan String (Find AL/AX/EAX)*	F2 A [111w]	x - - - x x x x x	4+5n	4+6n	4+5n	4+6n	b	h

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

INSTRUCTION	OPCODE	FLAGS									REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		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*		-	-	-	-	-	-	-	-	-					b	g,h,j,k,r
Within Segment	C3										10		10			
Within Segment Adding Immediate to SP	C2 ++										10		10			
Intersegment	CB										13	13	26	26		
Intersegment Adding Immediate to SP	CA ++										13	13	26	27		
Protected Mode: Different Privilege Level																
Intersegment													61	64		
Intersegment Adding Immediate to SP													61	64		
ROL *Rotate Left*		x	-	-	-	-	-	-	-	x					b	h
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*		x	-	-	-	-	-	-	-	x					b	h
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		
SAHF *Store AH in FLAGS*	9E	-	-	-	-	x	x	x	x	x	2		2			
SAL *Shift Left Arithmetic*		x	-	-	-	x	x	-	x	x					b	h
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*		x	-	-	-	x	x	-	x	x					b	h
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	6		
SBB *Integer Subtract with Borrow*		x	-	-	-	x	x	x	x	x					b	h
Register to Register	1 [10dw] [11 reg r/m]										1		1			
Register to Memory	1 [100w] [mod reg r/m]										3	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			

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
SCAS *Scan String*	A [111w]	x - - - x x x x x	5	5	5	5	b	h
SETB/SETNAE/SETC *Set Byte on Below/Not Above or Equal/Carry*		- - - - - - - - -						h
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*		- - - - - - - - -						h
To Register/Memory	0F 96 [mod 000 r/m]		2/2	2	2/2	2		
SETE/SETZ *Set Byte on Equal/Zero*		- - - - - - - - -						h
To Register/Memory	0F 94 [mod 000 r/m]		2/2	2	2/2	2		
SETL/SETNGE *Set Byte on Less/Not Greater or Equal*		- - - - - - - - -						h
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*		- - - - - - - - -						h
To Register/Memory	0F 9E [mod 000 r/m]		2/2	2	2/2	2		
SETNB/SETAE/SETNC *Set Byte on Not Below/ Above or Equal/Not Carry*	0F 93 [mod 000 r/m]	- - - - - - - - -	2/2	2	2/2	2		h
To Register/Memory								
SETNBE/SETA *Set Byte on Not Below or Equal/Above*		- - - - - - - - -						h
To Register/Memory	0F 97 [mod 000 r/m]		2/2	2	2/2	2		
SETNE/SETNZ *Set Byte on Not Equal/Not Zero*		- - - - - - - - -						h
To Register/Memory	0F 95 [mod 000 r/m]		2/2	2	2/2	2		

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
SETNL/SETGE *Set Byte on Not Less/Greater or Equal*		- - - - - - - - -						h
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*		- - - - - - - - -						h
To Register/Memory	0F 9F [mod 000 r/m]		2/2	2	2/2	2		
SETNO *Set Byte on Not Overflow*		- - - - - - - - -						h
To Register/Memory	0F 91 [mod 000 r/m]		2/2	2	2/2	2		
SETNP/SETPO *Set Byte on Not Parity/Parity Odd*		- - - - - - - - -						h
To Register/Memory	0F 9B [mod 000 r/m]		2/2	2	2/2	2		
SETNS *Set Byte on Not Sign*		- - - - - - - - -						h
To Register/Memory	0F 99 [mod 000 r/m]		2/2	2	2/2	2		
SETO *Set Byte on Overflow*		- - - - - - - - -						h
To Register/Memory	0F 90 [mod 000 r/m]		2/2	2	2/2	2		
SETP/SETPE *Set Byte on Parity/Parity Even*		- - - - - - - - -						h
To Register/Memory	0F 9A [mod 000 r/m]		2/2	2	2/2	2		
SETS *Set Byte on Sign*		- - - - - - - - -						h
To Register/Memory	0F 98 [mod 000 r/m]		2/2	2	2/2	2		
SGDT *Store GDT Register*		- - - - - - - - -					b,c	h
To Register/Memory	0F 01 [mod 000 r/m]		6	6	6	6		

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
SHL *Shift Left Logical*		x - - - x x - x x					b	h
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	7	1/3	7		
SHLD *Shift Left Double*		- - - - x x - x x						
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*		x - - - x x - x x					b	h
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	7	1/3	7		
SHRD *Shift Right Double*		- - - - x x - x x						
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*		- - - - - - - - -					b,c	h
To Register/Memory	0F 01 [mod 001 r/m]		6	6	6	6		
SLDT *Store LDT Register*		- - - - - - - - -					a	h
To Register/Memory	0F 00 [mod 000 r/m]				1/2	2		
SMSW *Store Machine Status Word*	0F 01 [mod 100 r/m]	- - - - - - - - -	1/2	2	1/2	2	b,c	h,l
STC *Set Carry Flag*	F9	- - - - - - - - 1	1		1			
STD *Set Direction Flag*	FD	- 1 - - - - - - -	1		1			
STI *Set Interrupt Flag*	FB	- - 1 - - - - - -	7		7			m

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		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]	- - - - - - - - - -	3	3	3	3	b	h
STR *Store Task Register*		- - - - - - - - - -					a	h
To Register/Memory	0F 00 [mod 001 r/m]				1/2	2		
SUB *Integer Subtract*		x - - - x x x x x					b	h
Register to Register	2 [10dw] [11 reg r/m]		1		1			
Register to Memory	2 [100w] [mod reg r/m]		3	5	3	5		
Memory to Register	2 [101w] [mod reg r/m]		3	5	3	5		
Immediate to Register/Memory	8 [00sw] [mod 101 r/m] #		1/3	5	1/3	5		
Immediate to Accumulator (short form)	2 [110w] #		1		1			
TEST *Test Bits*		0 - - - x x - x 0					b	h
Register/Memory and Register	8 [010w] [mod reg r/m]		1/3	5	1/3	5		
Immediate Data and Register/Memory	F [011w] [mod 000 r/m] #		1/3	5	1/3	5		
Immediate Data and Accumulator	A [100w] #		1		1			
VERR *Verify Read Access*		- - - - - x - - -					a	g,h,j,p
To Register/Memory	0F 00 [mod 100 r/m]				9/10	12		
VERW *Verify Write Access*		- - - - - x - - -					a	g,h,j,p
To Register/Memory	0F 00 [mod 101 r/m]				9/10	12		
WAIT *Wait Until FPU Not Busy*	9B	- - - - - - - - - -	5	5	5	5		
WBINVD *Write-Back and Invalidate Cache*	0F 09	- - - - - - - - - -	4		4			
XADD *Exchange and Add*		x - - - x x x x x						
Register1, Register2	0F C [000w] [11 reg2 reg1]		3		3			
Memory, Register	0F C [000w] [mod reg r/m]		6	6	6	6		

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

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT		PROTECTED MODE CLOCK COUNT		NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Cache Miss	Reg/ Cache Hit	Cache Miss	Real Mode	Protected Mode
XCHG *Exchange*		- - - - - - - - - -					b,f	f,h
Register/Memory with Register	8 [011w] [mod reg r/m]		3/4	4	3/4	4		
Register with Accumulator	9 [0 reg]		3		3			
XLAT *Translate Byte*	D7	- - - - - - - - - -	3	5	3	5		h
XOR *Boolean Exclusive OR*		0 - - - x x - x 0					b	h
Register to Register	3 [00dw] [11 reg r/m]		1		1			
Register to Memory	3 [000w] [mod reg r/m]		3	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	++ = 16-bit displacement	x = modified
+ = 8-bit displacement	+++ = 32-bit displacement (full)	- = unchanged

Instruction Notes for Instruction Set Summary

Notes a through c apply to Real Address Mode only:

a. This is a Protected Mode instruction. Attempted execution in Real Mode will result in exception 6 (invalid op-code).
b. Exception 13 fault (general protection) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum CS, DS, ES, FS, or GS segment limit (FFFFH). Exception 12 fault (stack segment limit violation or not present) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
c. This instruction may be executed in Real Mode. In Real Mode, its purpose is primarily to initialize the CPU for Protected Mode.

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

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

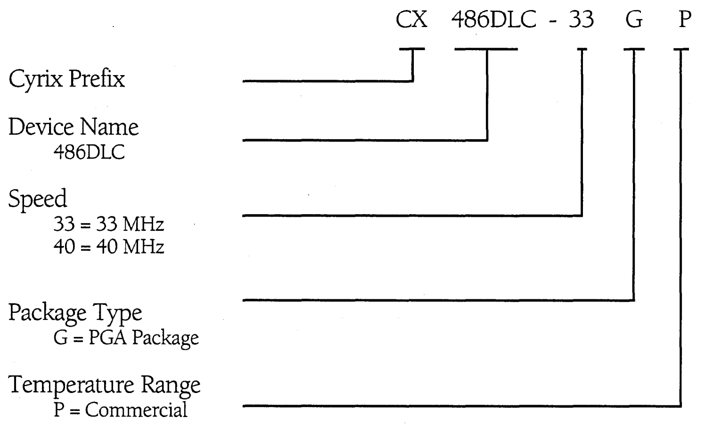
Notes h through r apply to Protected Virtual Address Mode only:

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

INDEX

INDEX

Ordering Information



The currently available Cyrix Cx486DLC part numbers are listed below:

PART NUMBER	DESCRIPTION
Cx486DLC-33GP	33 MHz, PGA Package
Cx486DLC-40GP	40 MHz, PGA Package

SALES OFFICES

US

Cyrix Corporation
2703 North Central Exprressway
Richardson, TX 75080
Tel: (214) 234-8387
Fax: (214) 699-9857

Europe

Cyrix Europe HQ
603 Delta Business Park
Welton Road
Swindon
SN5 7XF
UK
Tel: 44 (793) 417779
Fax: 44 (793) 417788

Taiwan & Hong Kong

ACCEL Technology Corp.
3F-1, 678 Tun-Hua S. Rd.
Taipei, Taiwan, R.O.C.
Tel: 886-02-755-2838
Fax: 886-02-708-0878

Japan

Cyrix KK
7F Nisso 11 Bldg, 2-3-4
Shin-Yokohama, Koh-Hoku-Ku
Yokohama, 222 Japan
Tel: 81-45-471-1661
Fax: 81-45-471-1666

Canada

Kaytronics
5800 Thimens Blvd.
Ville St. Laurent
Quebec H4S-1S5
Tel: (514) 745-6800
Fax: (514) 745-5858

Cyrix™
Advancing the Standards

2703 North Central Expressway ♦ Richardson, Texas 75080 ♦ (214) 234-8387

May 1992

Order Number 94076-01

Printed in the USA
Printed on Recycled Paper

Cyrix®
Advancing the Standards

2703 North Central Expressway Richardson, Texas 75080