Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

Embedded microprocessors are specialized computing chips designed to perform specific tasks within an embedded system. Unlike general-purpose microprocessors found in personal computers, embedded microprocessors are tailored for dedicated functions within larger systems, offering optimized performance, efficiency, and reliability. These microprocessors are integral to the operation of countless electronic devices, providing the computational power necessary for controlling processes, handling data, and managing communications.

Applications of Embedded - Microprocessors

Embedded microprocessors are utilized across a broad spectrum of applications, making them indispensable in

Details	
Product Status	Obsolete
Core Processor	80C286
Number of Cores/Bus Width	1 Core, 16-Bit
Speed	16MHz
Co-Processors/DSP	Math Engine; 80C287
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	68-LCC (J-Lead)
Supplier Device Package	68-PLCC (24.23x24.23)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=cs80c286-16

The 80C286 can be single-stepped using only the CPU clock. This state can be maintained as long as necessary. Single step clock information allows simple interface circuitry to provide critical information for system debug.

Static design also allows very low frequency operation (down to DC). In a power critical situation, this can provide low power operation since 80C286 power dissipation is directly related to operating frequency. As the system frequency is reduced, so is the operating power until, ultimately, with the clock stopped in phase two of the processor clock cycle, the 80C286 power requirement is the standby current (5mA maximum).

80C286 Base Architecture

The 80C86, 80C88, and 80C286 CPU family all contain the same basic set of registers, instructions, and addressing modes. The 80C286 processor is upwardly compatible with the 80C86 and 80C88 CPU's.

Register Set

The 80C286 base architecture has fifteen registers as shown in Figure 1. These registers are grouped into the following four categories.

GENERAL REGISTERS: Eight 16-bit general purpose registers used to contain arithmetic and logical operands. Four of these (AX, BX, CX and DX) can be used either in their entirety as 16-bit words or split into pairs of separate 8-bit registers.

SEGMENT REGISTERS: Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack and data. (For usage, refer to Memory Organization.)

BASE AND INDEX REGISTERS: Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode determines the specific registers used for operand address calculations.

STATUS AND CONTROL REGISTERS: Three 16-bit special purpose registers record or control certain aspects of the 80C286 processor state. These include the Flags register and Machine Status Word register shown in Figure 2, and the Instruction Pointer, which contains the offset address of the next sequential instruction to be executed.

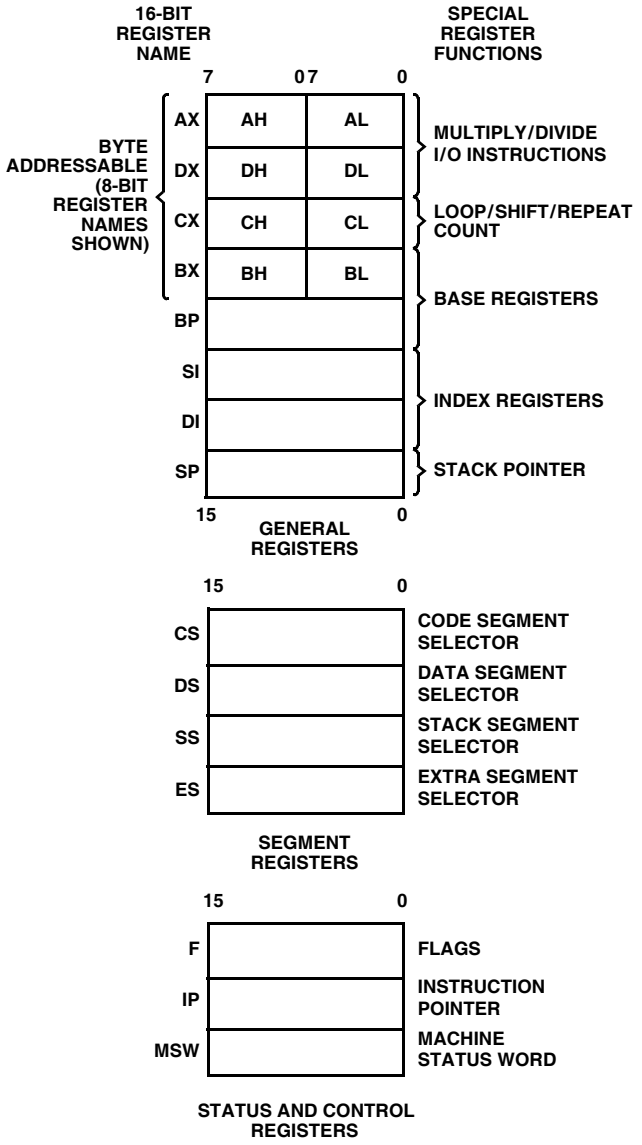


FIGURE 1. REGISTER SET

Flags Word Description

The Flags word (Flags) records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7 and 11) and controls the operation of the 80C286 within a given operating mode (bits 8 and 9). Flags is a 16-bit register. The function of the flag bits is given in Table 1.

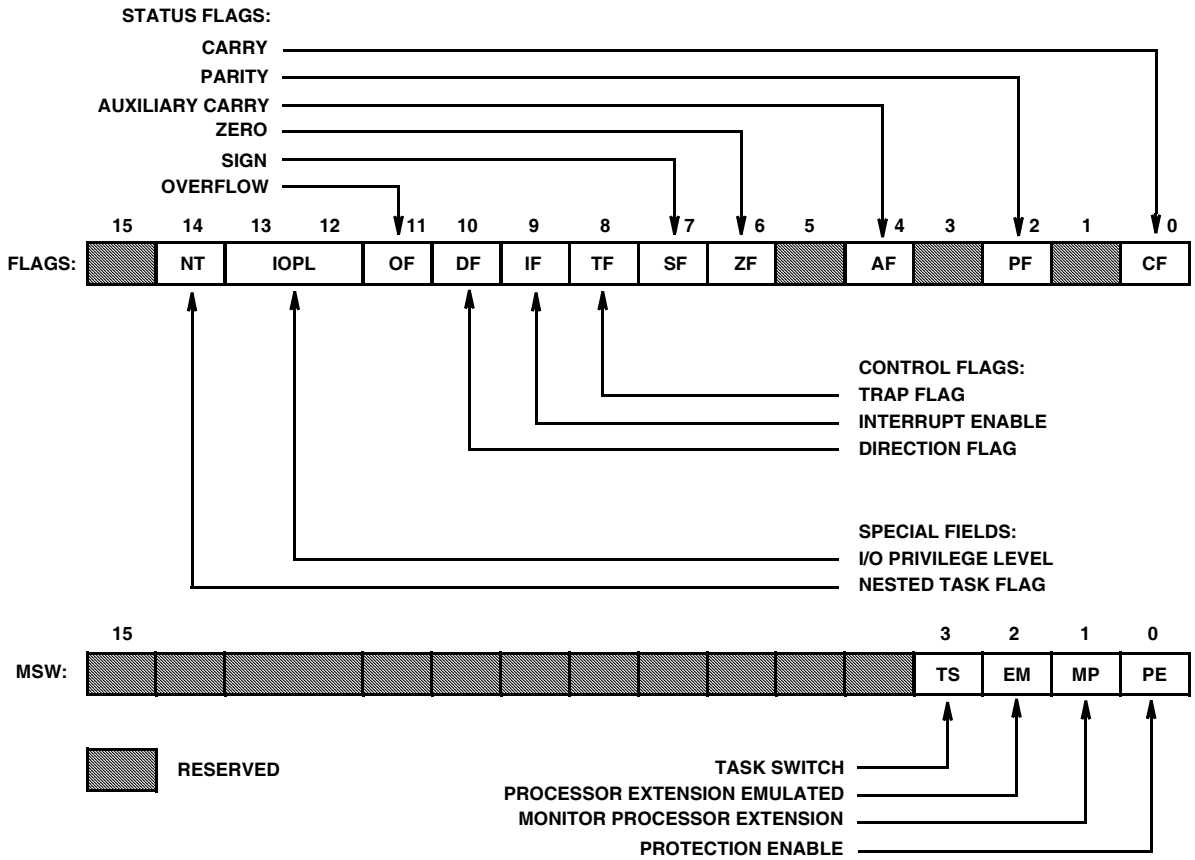


FIGURE 2. STATUS AND CONTROL REGISTER BIT FUNCTIONS

TABLE 1. FLAGS WORD BIT FUNCTIONS

BIT POSITION	NAME	FUNCTION
0	CF	Carry Flag - Set on high-order bit carry or borrow; cleared otherwise.
2	PF	Parity Flag - Set if low-order 8 bits of result contain an even number of 1 bits; cleared otherwise.
4	AF	Set on carry from or borrow to the low order four bits of AL; 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 if positive, 1 if negative).
11	OF	Overflow Flag - Set if result is a too-large positive number or a too-small negative number (excluding sign-bit) to fit in destination operand; cleared otherwise.
8	TF	Single Step Flag - Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-Enable Flag - When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag - Causes string instructions to auto decrement the appropriate index registers when set. Clearing DF causes auto increment.

Memory Organization

Memory is organized as sets of variable-length segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit segment selector and a 16-bit offset. The segment selector indicates the desired segment in memory. The offset component indicates the desired byte address within the segment. (See Figure 3).

All instructions that address operands in memory must specify the segment and the offset. For speed and compact instruction encoding, segment selectors are usually stored in the high speed segment registers. An instruction need specify only the desired segment register and offset in order to address a memory operand.

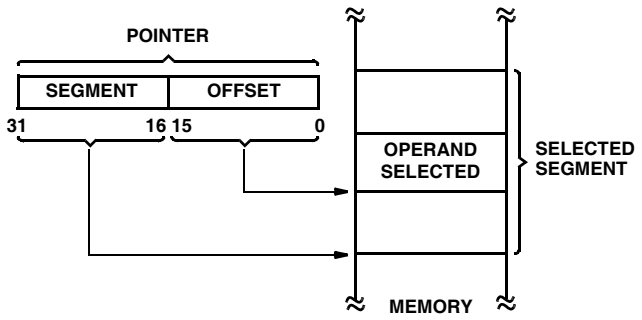


FIGURE 3. TWO COMPONENT ADDRESS

Most instructions need not explicitly specify which segment register is used. The correct segment register is automatically chosen according to the rules of Table 3. These rules follow the way programs are written (see Figure 4) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data and extra segments may coincide for simple programs. To access operands not residing in one of the four immediately available segments, a full 32-bit pointer or a new segment selector must be loaded.

TABLE 3. SEGMENT REGISTER SELECTION RULES

MEMORY REFERENCE NEEDED	SEGMENT REGISTER USED	IMPLICIT SEGMENT SELECTION RULE
Instructions	Code (CS)	Automatic with instruction prefetch
Stack	Stack (SS)	All stack pushes and pops. Any memory reference which uses BP as a base register.
Local Data	Data (DS)	All data references except when relative to stack or string destination
External (Global) Data	Extra (ES)	Alternate data segment and destination of string operation

Addressing Modes

The 80C286 provides a total of eight addressing modes for instructions to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

REGISTER OPERAND MODE: The operand is located in one of the 8 or 16-bit general registers.

IMMEDIATE OPERAND MODE: The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: segment selector and offset. The segment selector is supplied by a segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset is calculated by summing any combination of the following three address elements:

the **displacement** (an 8 or 16-bit immediate value contained in the instruction)

the **base** (contents of either the BX or BP base registers)

the **index** (contents of either the SI or DI index registers)

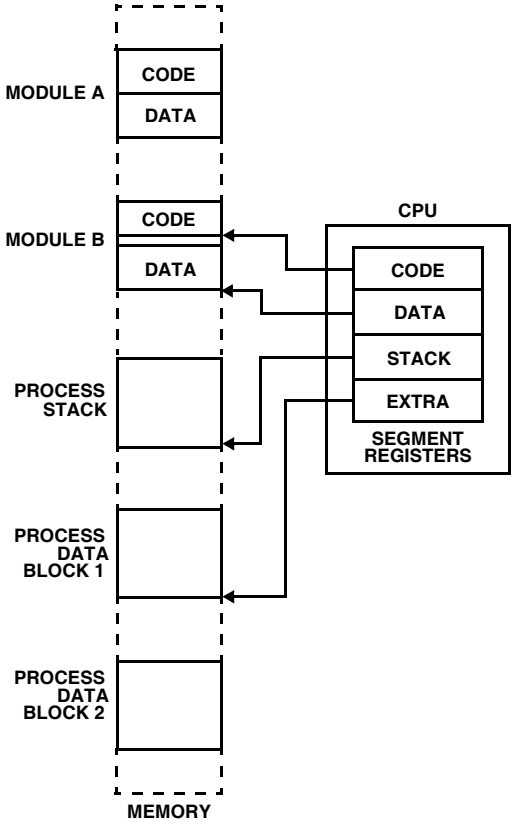


FIGURE 4. SEGMENTED MEMORY HELPS STRUCTURE SOFTWARE

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

DIRECT MODE: The operand's offset is contained in the instruction as an 8 or 16-bit displacement element.

REGISTER INDIRECT MODE: The operand's offset is in one of the registers SI, DI, BX or BP.

BASED MODE: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of a base register (BX or BP).

INDEXED MODE: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of an index register (SI or DI).

BASED INDEXED MODE: The operand's offset is the sum of the contents of a base register and an index register.

BASED INDEXED MODE WITH DISPLACEMENT: The operand's offset is the sum of a base register's contents, an index register's contents, and an 8 or 16-bit displacement.

Data Types

The 80C286 directly supports the following data types:

Integer: A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32 and 64-bit integers are supported using the 80287 Numeric Data Processor.

Ordinal: An unsigned binary numeric value contained in an 8-bit byte or 16-bit word.

Pointer: A 32-bit quantity, composed of a segment selector component and an offset component. Each component is a 16-bit word.

String: A contiguous sequence of bytes or words. A string may contain from 1 byte to 64K bytes.

ASCII: A byte representation of alphanumeric and control characters using the ASCII standard of character representation.

BCD: A byte (unpacked) representation of the decimal digits 0-9.

Packed BCD: A byte (packed) representation of two decimal digits 0-9 storing one digit in each nibble of the byte.

Floating Point: A signed 32, 64 or 80-bit real number representation. (Floating point operands are supported using the 80287 Numeric Processor extension).

Figure 5 graphically represents the data types supported by the 80C286.

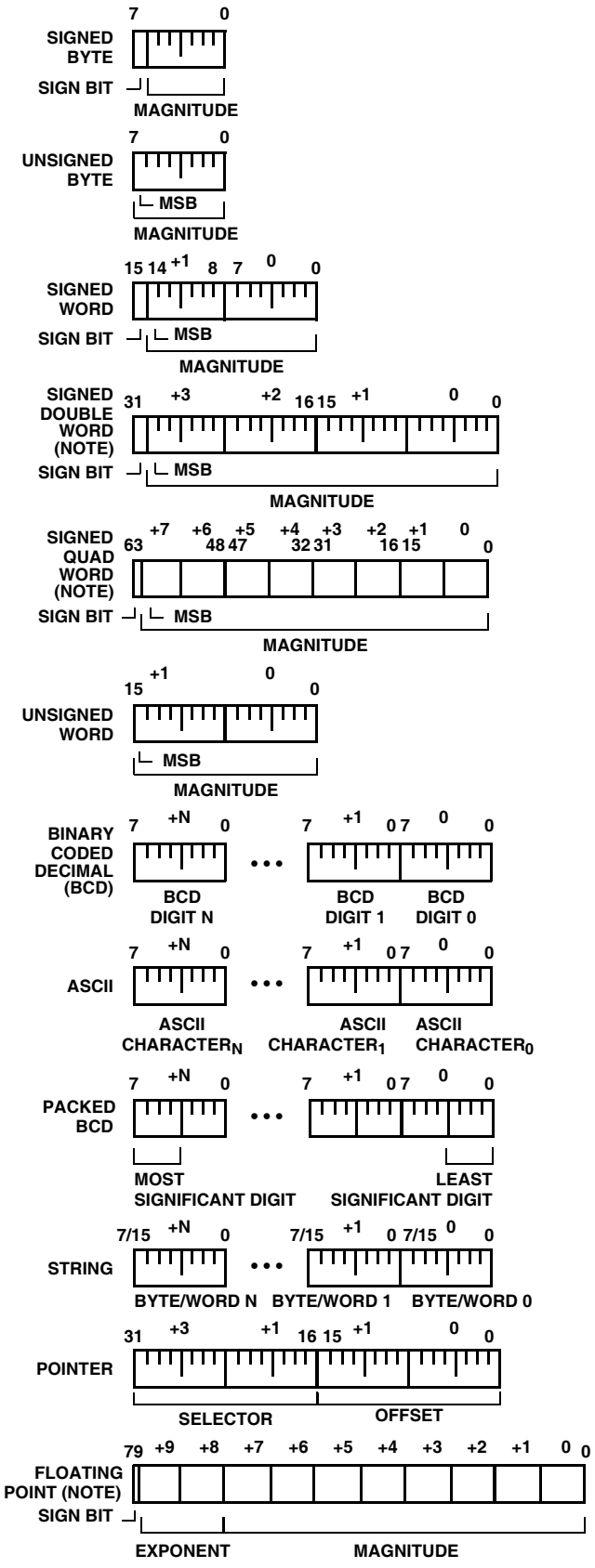


FIGURE 5. 80C286 SUPPORTED DATA TYPES

NOTE: Supported by 80C286/80C287 Numeric Data Processor Configuration

TABLE 4. INTERRUPT VECTOR ASSIGNMENTS

FUNCTION	INTERRUPT NUMBER	RELATED INSTRUCTIONS	DOES RETURN ADDRESS POINT TO INSTRUCTION CAUSING EXCEPTION?
Divide Error Exception	0	DIV, IDIV	Yes
Single Step Interrupt	1	All	
NMI Interrupt	2	INT 2 or NMI Pin	
Breakpoint Interrupt	3	INT 3	
INTO Detected Overflow Exception	4	INTO	No
BOUND Range Exceeded Exception	s	BOUND	Yes
Invalid Opcode Exception	6	Any Undefined Opcode	Yes
Processor Extension Not Available Exception	7	ESC or WAIT	Yes
Reserved - Do Not Use	8 - 15		
Processor Extension Error Interrupt	16	ESC or WAIT	
Reserved	17 - 31		
User Defined	32 - 255		

I/O Space

The I/O space consists of 64K 8-bit ports, 32K 16-bit ports, or a combination of the two. I/O instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}-A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Flags) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable. Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition which prevents further instruction processing is detected while attempting to execute an instruction. The return address from an exception will always point to the instruction causing the exception and include any leading instruction prefixes.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0-31, some of which are used for instruction exceptions, are reserved. For each interrupt, an 8-bit vector must be supplied to the 80C286 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Nonmaskable hardware interrupts use a predefined internally supplied vector.

Maskable Interrupt (INTR)

The 80C286 provides a maskable hardware interrupt request pin, INTR. Software enables this input by setting the interrupt flag bit (IF) in the flag word. All 224 user-defined interrupt sources can share this input, yet they can retain separate interrupt handlers. An 8-bit vector read by the CPU during the interrupt acknowledge sequence (discussed in System Interface section) identifies the source of the interrupt.

The processor automatically disables further maskable interrupts internally by resetting the IF as part of the response to an interrupt or exception. The saved flag word will reflect the enable status of the processor prior to the interrupt. Until the flag word is restored to the flag register, the interrupt flag will be zero unless specifically set. The interrupt return instruction includes restoring the flag word, thereby restoring the original status of IF.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt input (NMI) is also provided. NMI has higher priority than INTR. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed.

While executing the NMI servicing procedure, the 80C286 will service neither further NMI requests, INTR requests, nor the processor extension segment overrun interrupt until an interrupt return (IRET) instruction is executed or the CPU is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. IF is cleared at the beginning of an NMI interrupt to inhibit INTR interrupts.

TABLE 8. RECOMMENDED MSW ENCODINGS FOR PROCESSOR EXTENSION CONTROL

TS	MP	EM	RECOMMENDED USE	INSTRUCTION CAUSING EXCEPTION 7
0	0	0	Initial encoding after RESET. 80C286 operation is identical to 80C86/88.	None
0	0	1	No processor extension is available. Software will emulate its function.	ESC
1	0	1	No processor extension is available. Software will emulate its function. The current processor extension context may belong to another task.	ESC
0	1	0	A processor extension exists.	None
1	1	0	A processor extension exists. The current processor extension context may belong to another task. The exception 7 on WAIT allows software to test for an error pending from a previous processor extension operation.	ESC or WAIT

TABLE 9. REAL ADDRESS MODE ADDRESSING INTERRUPTS

FUNCTION	INTERRUPT NUMBER	RELATED INSTRUCTIONS	RETURN ADDRESS BEFORE INSTRUCTION
Interrupt table limit too small exception	8	INT vector is not within table limit	Yes
Processor extension segment overrun interrupt	9	ESC with memory operand extending beyond offset FFFF(H)	No
Segment overrun exception	13	Word memory reference with offset = FFFF(H) or an attempt to execute past the end of a segment	Yes

80C286 Real Address Mode

The 80C286 executes a fully upward-compatible superset of the 80C86 instruction set in real address mode. In real address mode the 80C286 is object code compatible with 80C86 and 80C88 software. The real address mode architecture (registers and addressing modes) is exactly as described in the 80C286 Base Architecture section of this Functional Description.

Memory Size

Physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A_0 through A_{19} and $\overline{BHE}$. A_{20} through A_{23} should be ignored.

Memory Addressing

In real address mode physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pin A_0 through A_{19} and $\overline{BHE}$. Address bits A_{20}-A_{23} may not always be zero in real mode. A_{20}-A_{23} should not be used by the system while the 80C286 is operating in Real Mode.

The selector portion of a pointer is interpreted as the upper 16-bits of a 20-bit segment address. The lower four bits of the 20-bit segment address are always zero. Segment addresses, therefore, begin on multiples of 16 bytes. See Figure 6 for a graphic representation of address information.

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

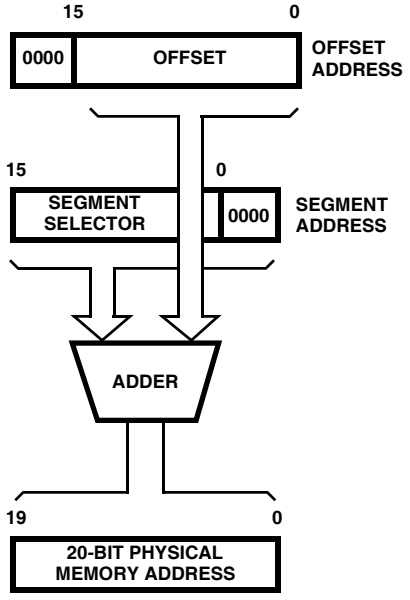


FIGURE 6. 80C286 REAL ADDRESS MODE ADDRESS CALCULATION

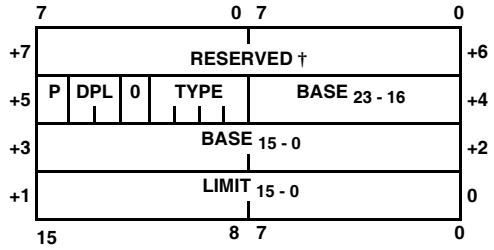
Data segments (S = 1, E = 0) may be either read-only or read-write as controlled by the W bit of the access rights byte. Read-only (W = 0) data segments may not be written into. Data segments may grow in two directions, as determined by the Expansion Direction (ED) bit: upwards (ED = 0) for data segments, and downwards (ED = 1) for a segment containing a stack. The limit field for a data segment descriptor is interpreted differently depending on the ED bit (see Table 10).

A code segment (S = 1, E = 1) may be execute-only or execute/read as determined by the Readable (R) bit. Code segments may never be written into and execute-only code segments (R = 0) may not be read. A code segment may also have an attribute called conforming (C). A conforming code segment may be shared by programs that execute at different privilege levels. The DPL of a conforming code segment defines the range of privilege levels at which the segment may be executed (refer to privilege discussion below). The limit field identifies the last byte of a code segment.

System Segment Descriptors (S = 0, Type = 1-3)

In addition to code and data segment descriptors, the protected mode 80C286 defines System Segment Descriptors. These descriptors define special system data segments which contain a table of descriptors (Local Descriptor Table Descriptor) or segments which contain the execution state of a task (Task State Segment Descriptor).

Table 11 gives the formats for the special system data segment descriptors. The descriptors contain a 24-bit base address of the segment and a 16-bit limit. The access byte defines the type of descriptor, its state and privilege level. The descriptor contents are valid and the segment is in physical memory if P = 1. If P = 0, the segment is not valid. The DPL field is only used in Task State Segment descriptors and indicates the privilege level at which the descriptor may be used (see Privilege). Since the Local Descriptor Table descriptor may only be used by a special privileged instruction, the DPL field is not used. Bit 4 of the access byte is 0 to indicate that it is a system control descriptor. The type field specifies the descriptor type as indicated in Table 11.



† MUST BE SET TO 0 FOR COMPATIBILITY WITH FUTURE UPGRADES

FIGURE 10. SYSTEM SEGMENT DESCRIPTOR

TABLE 11. SYSTEM SEGMENT DESCRIPTOR FORMAT FIELDS

NAME	VALUE	DESCRIPTION
TYPE	1	Available Task State Segment (TSS)
	2	Local Descriptor Table
	3	Busy Task State Segment (TSS)
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid
DPL	0-3	Descriptor Privilege Level
BASE	24-Bit Number	Base Address of special system data segment in real memory
LIMIT	16-Bit Number	Offset of last byte in segment

Gate Descriptors (S = 0, Type = 4-7)

Gates are used to control access to entry points within the target code segment. The gate descriptors are call gates, task gates, interrupt gates and trap gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the CPU to automatically perform protection checks and control entry point of the destination. Call gates are used to change privilege levels (see Privilege), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines. The interrupt gate disables interrupts (resets IF) while the trap gate does not.

Table 12 shows the format of the gate descriptors. The descriptor contains a destination pointer that points to the descriptor of the target segment and the entry point offset. The destination selector in an interrupt gate, trap gate, and call gate must refer to a code segment descriptor. These gate descriptors contain the entry point to prevent a program from constructing and using an illegal entry point. Task gates may only refer to a task state segment. Since task gates invoke a task switch, the destination offset is not used in the task gate.

Exception 13 is generated when the gate is used if a destination selector does not refer to the correct descriptor type. The word count field is used in the call gate descriptor to indicate the number of parameters (0-31 words) to be automatically copied from the caller's stack to the stack of the called routine when a control transfer changes privilege levels. The word count field is not used by any other gate descriptor.

The access byte format is the same for all descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (refer to privilege discussion below). Bit 4 must equal 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Table 12.

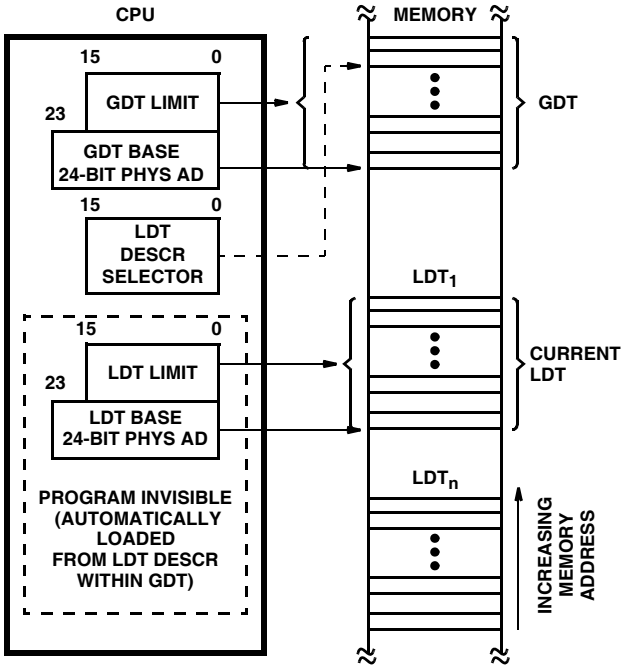
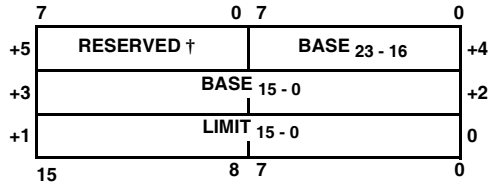


FIGURE 14. LOCAL AND GLOBAL DESCRIPTOR TABLE DEFINITION

The LGDT and LLDT instructions load the base and limit of the global and local descriptor tables. LGDT and LLDT are privileged, i.e. they may only be executed by trusted programs operating at level 0. The LGDT instruction loads a six byte field containing the 16-bit table limit and 24-bit physical base address of the Global Descriptor Table as shown in Figure 15. The LDT instruction loads a selector which refers to a Local Descriptor Table descriptor containing the base address and limit for an LDT, as shown in Table 11.



† MUST BE SET TO 0 FOR COMPATIBILITY WITH FUTURE UPGRADES

FIGURE 15. GLOBAL DESCRIPTOR TABLE AND INTERRUPT DESCRIPTOR TABLE DATA TYPE

Interrupt Descriptor Table

The protected mode 80C286 has a third descriptor table, called the Interrupt Descriptor Table (IDT) (see Figure 16), used to define up to 256 interrupts. It may contain only task gates, interrupt gates and trap gates. The IDT (Interrupt Descriptor Table) has a 24-bit physical base and 16-bit limit register in the CPU. The privileged LIDT instruction loads these registers with a six byte value of identical form to that of the LGDT instruction (see Figure 16 and Protected Mode Initialization).

References to IDT entries are made via INT instructions, external interrupt vectors, or exceptions. The IDT must be at least 256 bytes in size to allocate space for all reserved interrupts.

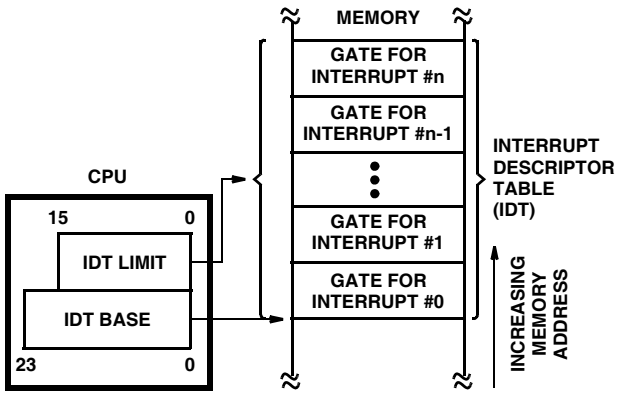
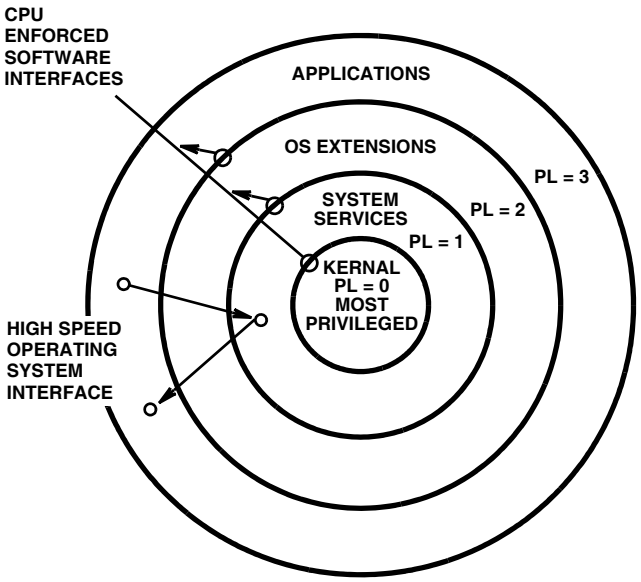


FIGURE 16. INTERRUPT DESCRIPTOR TABLE DEFINITION

Privilege

The 80C286 has a four-level hierarchical privilege system which controls the use of privileged instructions and access to descriptors (and their associated segments) within a task. Four-level privilege, as shown in Figure 17, is an extension of the users/supervisor mode commonly found in minicomputers. The privilege levels are numbered 0 through 3. Level 0 is the most privileged level. Privilege levels provide protection within a task. (Tasks are isolated by providing private LDT's for each task.) Operating system routines, interrupt handlers, and other system software can be included and protected within the virtual address space of each task using the four levels of privilege. Each task in the system has a separate stack for each of its privilege levels.

Tasks, descriptors, and selectors have a privilege level attribute that determines whether the descriptor may be used. Task privilege affects the use of instructions and descriptors. Descriptor and selector privilege only affect access to the descriptor.



NOTE: PL becomes numerically lower as privilege level increases.

FIGURE 17. HIERARCHICAL PRIVILEGE LEVELS

function of the IRET instruction. If NT = 0, the IRET instruction performs the regular current task by popping values off the stack; when NT = 1, IRET performs a task switch operation back to the previous task.

When a CALL, JMP, or INT instruction initiates a task switch, the old (except for case of JMP) and new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. NT may also be set or cleared by POPF or IRET instructions.

The task state segment is marked busy by changing the descriptor type field from Type 1 to Type 3. Use of a selector that references a busy task state segment causes Exception 13.

Processor Extension Context Switching

The context of a processor extension is not changed by the task switch operation. A processor extension context need only be changed when a different task attempts to use the processor extension (which still contains the context of a previous task). The 80C286 detects the first use of a processor extension after a task switch by causing the processor extension not present exception (7). The interrupt handler may then decide whether a context change is necessary.

Whenever the 80C286 switches tasks, it sets the Task Switched (TS) bit of the MSW. TS indicates that a processor extension context may belong to a different task than the current one. The processor extension not present exception (7) will occur when attempting to execute an ESC or WAIT instruction if TS = 1 and a processor extension is present (MP = 1 in MSW).

Pointer Testing Instructions

The 80C286 provides several instructions to speed pointer testing and consistency checks for maintaining system integrity (see Table 18). These instructions use the memory management hardware to verify that a selector value refers to an appropriate segment without risking an exception. A condition flag (ZF) indicates whether use of the selector or segment will cause an exception.

Double Fault and Shutdown

If two separate exceptions are detected during a single instruction execution, the 80C286 performs the double fault exception (8). If an exception occurs during processing of the double fault exception, the 80C286 will enter shutdown. During shutdown no further instructions or exceptions are processed. Either NMI (CPU remains in protected mode) or RESET (CPU exits protected mode) can force the 80C286 out of shutdown. Shutdown is externally signalled via a HALT bus operation with A_1 LOW.

Protected Mode Initialization

The 80C286 initially executes in real address mode after RESET. To allow initialization code to be placed at the top of physical memory. A_{23-20} will be HIGH when the 80C286 performs memory references relative to the CS register until CS is changed. A_{23-20} will be zero for references to the DS, ES, or SS segments. Changing CS in real address mode will force A_{23-20} LOW whenever CS is used again. The initial CS:IP value of F000:FFF0 provides 64K bytes of code space for initialization code without changing CS.

Protected mode operation requires several registers to be initialized. The GDT and IDT base registers must refer to a valid GDT and IDT. After executing the LMSW instruction to set PE, the 80C286 must immediately execute an intrasegment JMP instruction to clear the instruction queue of instructions decoded in real address mode.

To force the 80C286 CPU registers to match the initial protected mode state assumed by software, execute a JMP instruction with a selector referring to the initial TSS used in the system. This will load the task register, local descriptor table register, segment registers and initial general register state. The TR should point at a valid TSS since any task switch operation involves saving the current task state.

TABLE 18. 80C286 POINTER TEST INSTRUCTIONS

INSTRUCTION	OPERANDS	FUNCTION
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed by ARPL.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

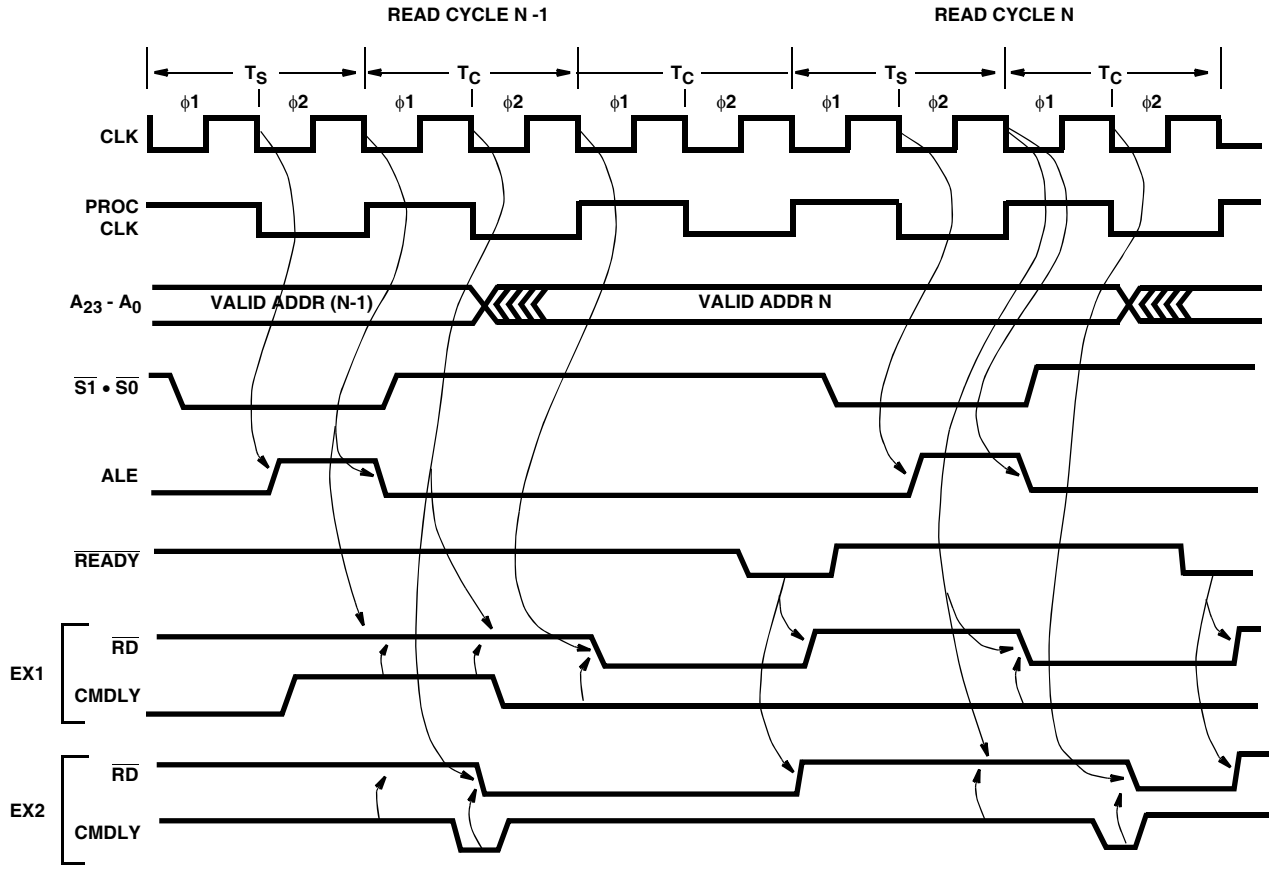


FIGURE 23. CMDLY CONTROLS THE LEADING EDGE OF COMMAND SIGNAL

Bus Cycle Termination

At maximum transfer rates, the 80C286 bus alternates between the status and command states. The bus status signals become inactive after T_S so that they may correctly signal the start of the next bus operation after the completion of the current cycle. No external indication of T_C exists on the 80C286 local bus. The bus master and bus controller enter T_C directly after T_S and continue executing T_C cycles until terminated by the assertion of $\overline{READY}$.

$\overline{READY}$ Operation

The current bus master and 82C288 bus controller terminate each bus operation simultaneously to achieve maximum bus operation bandwidth. Both are informed in advance by $\overline{READY}$ active (open-collector output from 82C284) which identifies the last T_C cycle of the current bus operation. The bus master and bus controller must see the same sense of the $\overline{READY}$ signal, thereby requiring $\overline{READY}$ to be synchronous to the system clock.

Synchronous Ready

The 82C284 clock generator provides $\overline{READY}$ synchronization from both synchronous and asynchronous sources (see Figure 24). The synchronous ready input ($\overline{SRDY}$) of the clock generator is sampled with the falling edge of CLK at the end of phase 1 of each T_C. The state of $\overline{SRDY}$ is then broadcast to the bus master and bus controller via the $\overline{READY}$ output line.

Asynchronous Ready

Many systems have devices or subsystems that are asynchronous to the system clock. As a result, their ready outputs cannot be guaranteed to meet the 82C284 $\overline{SRDY}$ setup and hold time requirements. But the 82C284 asynchronous ready input ($\overline{ARDY}$) is designed to accept such signals. The $\overline{ARDY}$ input is sampled at the beginning of each T_C cycle by 82C284 synchronization logic. This provides one system CLK cycle time to resolve its value before broadcasting it to the bus master and bus controller.

$\overline{ARDY}$ or $\overline{ARDYEN}$ must be HIGH at the end of T_S. $\overline{ARDY}$ cannot be used to terminate the bus cycle with no wait states.

Each ready input of the 82C284 has an enable pin ($\overline{SRDYEN}$ and $\overline{ARDYEN}$) to select whether the current bus operation will be terminated by the synchronous or asynchronous ready. Either of the ready inputs may terminate a bus operation. These enable inputs are active low and have the same timing as their respective ready inputs. Address decode logic usually selects whether the current bus operation should be terminated by $\overline{ARDY}$ or $\overline{SRDY}$.

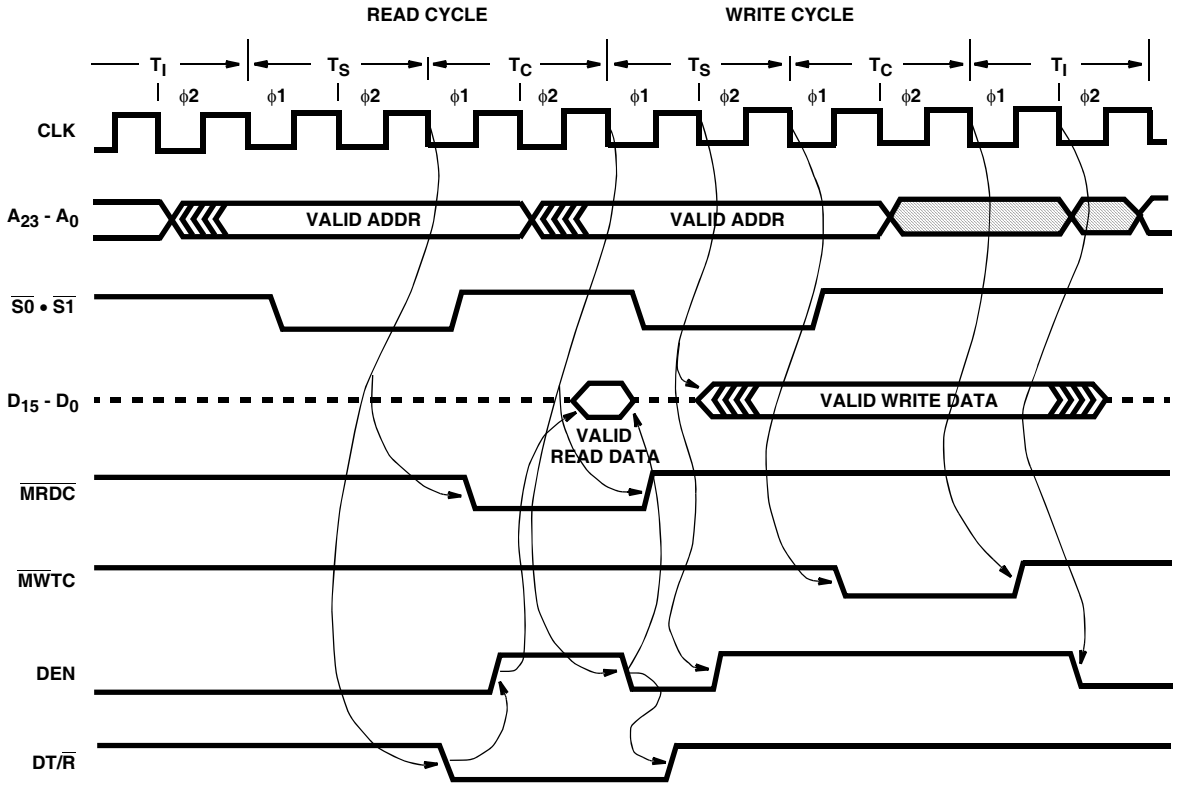


FIGURE 25. BACK TO BACK READ-WRITE CYCLE

Local Bus Usage Priorities

The 80C286 local bus is shared among several internal units and external HOLD requests. In case of simultaneous requests, their relative priorities are:

(Highest) Any transfers which assert $\overline{LOCK}$ either explicitly (via the LOCK instruction prefix) or implicitly (i.e. some segment descriptor accesses, an interrupt acknowledge sequence, or an XCHG with memory).

The second of the two byte bus operations required for an odd aligned word operand.

The second or third cycle of a processor extension data transfer.

Local bus request via HOLD input.

Processor extension data operand transfer via PEREQ input.

Data transfer performed by EU as part of an instruction.

(Lowest) An instruction prefetch request from BU. The EU will inhibit prefetching two processor clocks in advance of any data transfers to minimize waiting by the EU for a prefetch to finish.

Halt or Shutdown Cycles

The 80C286 externally indicates halt or shutdown conditions as a bus operation. These conditions occur due to a HLT instruction or multiple protection exceptions while attempting to execute one instruction. A halt or shutdown bus operation is signalled when $\overline{S_1}$, $\overline{S_0}$ and COD/$\overline{INTA}$ are LOW and M/$\overline{IO}$ is HIGH. A_1 HIGH indicates halt, and A_1 LOW indicates shutdown. The 82C288 bus controller does not issue ALE, nor is $\overline{READY}$ required to terminate a halt or shutdown bus operation.

During halt or shutdown, the 80C286 may service PEREQ or HOLD requests. A processor extension segment overrun during shutdown will inhibit further service of PEREQ. Either NMI or RESET will force the 80C286 out of either halt or shutdown. An INTR, if interrupts are enabled, or a processor extension segment overrun exception will also force the 80C286 out of halt.

System Configurations

The versatile bus structure of the 80C286 micro-system, with a full complement of support chips, allows flexible configuration of a wide range of systems. The basic configuration, shown in Figure 30, is similar to an 80C86 maximum mode system. It includes the CPU plus an 82C59A interrupt controller, 82C284 clock generator, and the 82C288 Bus Controller. The 80C86 latches (82C82 and 82C83H) and transceivers (82C86H and 82C87H) may be used in an 80C286 microsystem.

As indicated by the dashed lines in Figure 30, the ability to add processor extensions is an integral feature of 80C286 based microsystems. The processor extension interface allows external hardware to perform special functions and transfer data concurrent with CPU execution of other instructions. Full system integrity is maintained because the 80C286 supervises all data transfers and instruction execution for the processor extension.

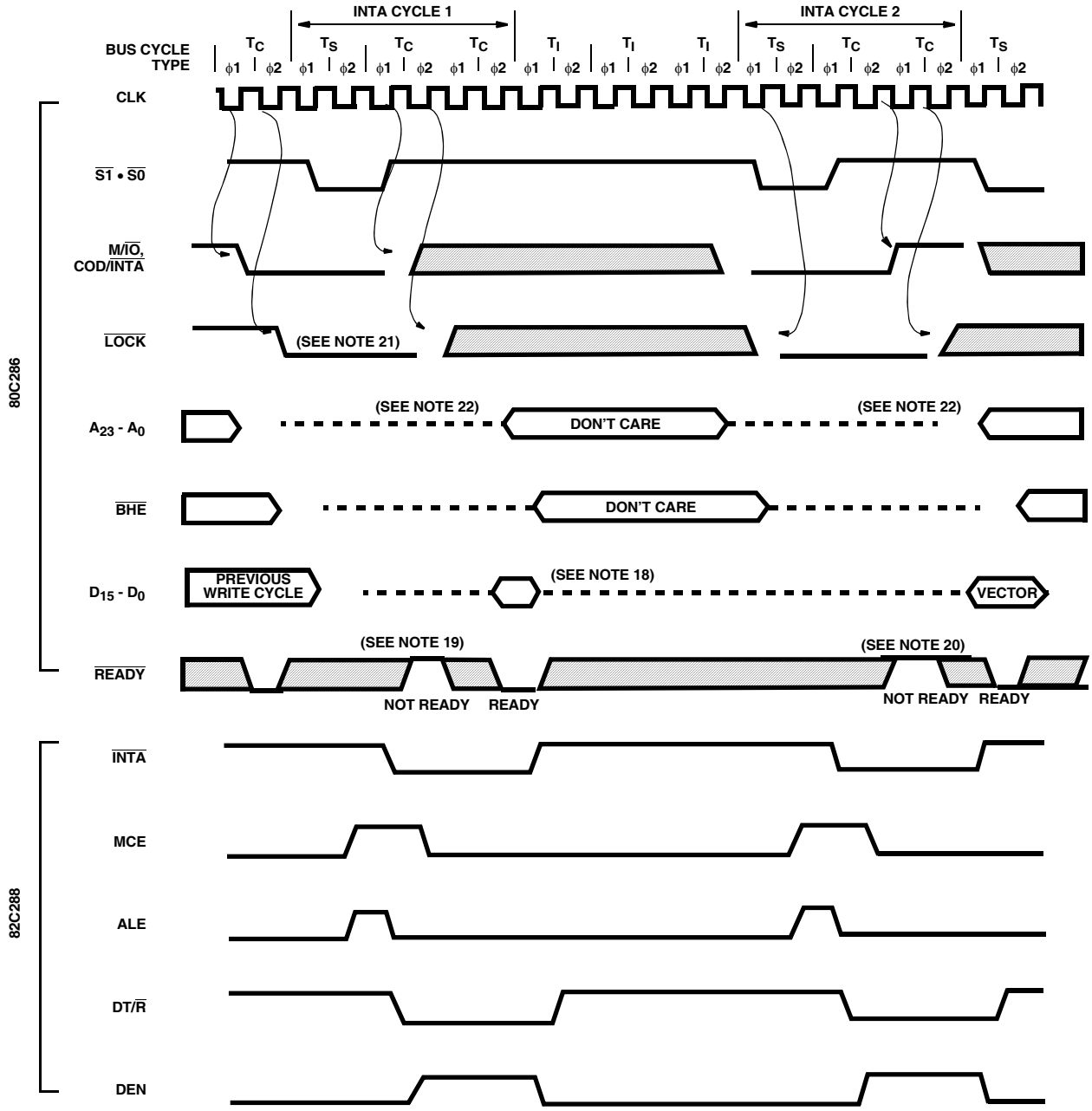
An 80C286 system which includes the 80287 numeric processor extension (NPX) uses this interface. The 80C286/80287 system has all the instructions and data types of an 80C86 or 80C88 with 8087 numeric processor extension. The 80287 NPX can perform numeric calculations and data transfers concurrently with CPU program execution. Numerics code and data have the same integrity as all other information protected by the 80C286 protection mechanism.

The 80C286 can overlap chip select decoding and address propagation during the data transfer for the previous bus operation. This information is latched into the 82C82/83H's by ALE during the middle of a T_S cycle. The latched chip select and address information remains stable during the bus operation while the next cycle's address is being decoded and propagated into the system. Decode logic can be implemented with a high speed PROM or PAL.

The optional decode logic shown in Figure 30 takes advantage of the overlap between address and data of the 80C286 bus cycle to generate advanced memory and I/O select signals. This minimizes system performance degradation caused by address propagation and decode delays. In addition to selecting memory and I/O, the advanced selects may be used with configurations supporting local and system buses to enable the appropriate bus interface for each bus cycle. The COD/$\overline{INTA}$ and M/$\overline{IO}$ signals are applied to the decode logic to distinguish between interrupt, I/O, code, and data bus cycles.

By adding the 82289 bus arbiter chip the 80C286 provides a Multibus system bus interface as shown in Figure 31. The ALE output of the 82C288 for the Multibus bus is connected to its CMDLY input to delay the start of commands one system CLK as required to meet Multibus address and write data setup times. This arrangement will add at least one extra T_C state to each bus operation which uses the Multibus.

A second 82C288 bus controller and additional latches and transceivers could be added to the local bus of Figure 31. This configuration allows the 80C286 to support an on-board bus for local memory and peripherals, and the Multibus for system bus interfacing.



NOTES:

18. Data is ignored.
19. First INTA cycle should have at least one wait state inserted to meet 82C59A minimum INTA pulse width.
20. Second INTA cycle must have at least one wait state inserted since the CPA will not drive A_{23}-A_0, $\overline{BHE}$, and $\overline{LOCK}$ until after the first T_C state. The CPU imposed one/clock delay prevents has contention between cascade address buffer being disabled by MCE $\downarrow$ and address outputs.
21. Without the wait state, the 80C286 address will not be valid for a memory cycle started immediately after the second INTA cycle. The 82C59A also requires one wait state for minimum INTA pulse width.
22. $\overline{LOCK}$ is active for the first INTA cycle to prevent the 82289 from releasing the bus between INTA cycles in a multi-master system. $\overline{LOCK}$ is also active for the second INTA cycle.
23. A_{23}-A_0 exits three-state OFF during $\phi2$ of the second T_C in the INTA cycle.

FIGURE 29. INTERRUPT ACKNOWLEDGE SEQUENCE

Waveforms

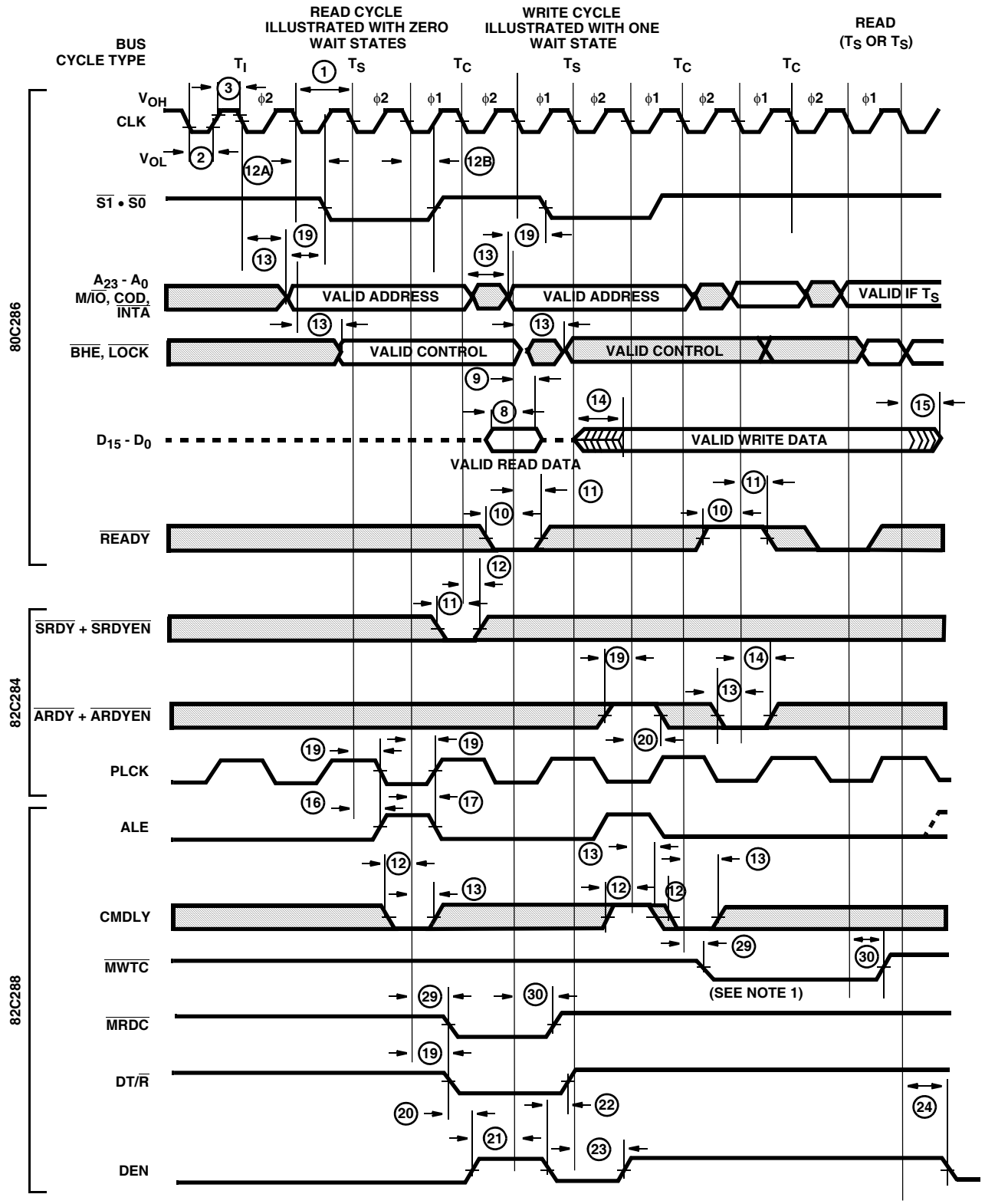
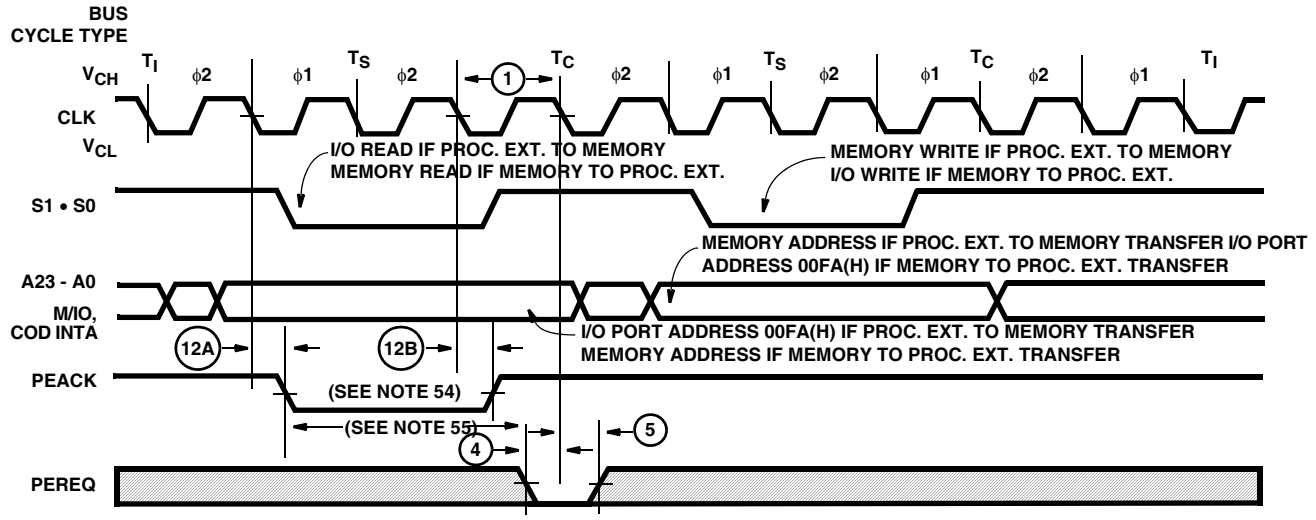


FIGURE 34. MAJOR CYCLE TIMING

NOTE: The modified timing is due to the CMDLY signal being active.

Waveforms (Continued)



ASSUMING WORD-ALIGNED MEMORY OPERAND. IF ODD ALIGNED, 80C286 TRANSFERS TO/FROM MEMORY BYTE-AT-A-TIME WITH TWO MEMORY CYCLES.

FIGURE 38. 80C286 PEREQ/$\overline{\text{PEACK}}$ TIMING FOR ONE TRANSFER ONLY

NOTES:

54. $\overline{\text{PEACK}}$ always goes active during the first bus operation of a processor extension data operand transfer sequence. The first bus operation will be either a memory read at operand address or I/O read at port address 00FA(H).

55. To prevent a second processor extension data operand transfer, the worst case maximum time (Shown above) is 3 x ① - $12A_{MAX}$ - $④_{MIN}$. The actual configuration dependent, maximum time is: 3 x ① - $12A_{MAX}$ - $④_{MIN}$ + N x 2 x ①. N is the number of extra T_C states added to either the first or second bus operation of the processor extension data operand transfer sequence.

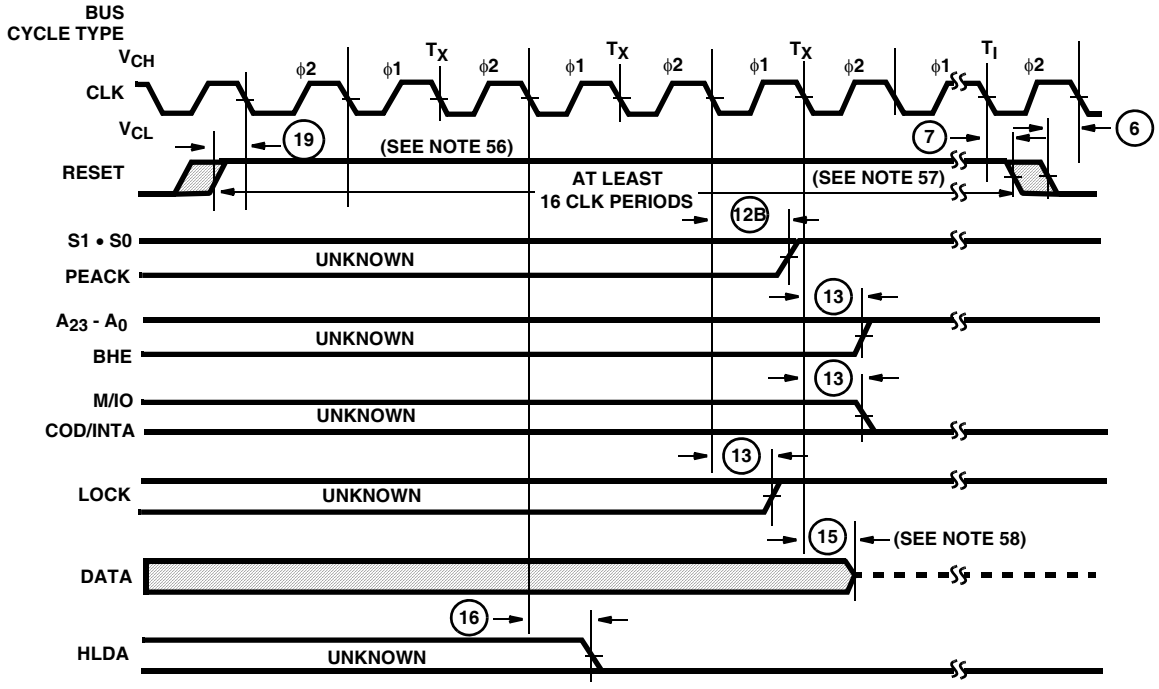


FIGURE 39. INITIAL 80C286 PIN STATE DURING RESET

NOTES:

56. Setup time for RESET ↑ may be violated with the consideration that $\phi 1$ of the processor clock may begin one system CLK period later.

57. Setup and hold times for RESET ↓ must be met for proper operation, but RESET ↓ may occur during $\phi 1$ or $\phi 2$.

58. The data bus is only guaranteed to be in a high impedance state at the time shown.

Real Address Mode Only

1. This is a protected mode instruction. Attempted execution in real address mode will result in an undefined opcode exception (6).
2. A segment overrun exception (13) will occur if a word operand references at offset FFFF(H) is attempted.
3. This instruction may be executed in real address mode to initialize the CPU for protected mode.
4. The IOPL and NT fields will remain 0.
5. Processor extension segment overrun interrupt (9) will occur if the operand exceeds the segment limit.

Either Mode

6. An exception may occur, depending on the value of the operand.
7. $\overline{\text{LOCK}}$ is automatically asserted regardless of the presence or absence of the $\overline{\text{LOCK}}$ instruction prefix.
8. $\overline{\text{LOCK}}$ does not remain active between all operand transfers.

Protected Virtual Address Mode Only

9. A general protection exception (13) will occur if the memory operand cannot be used due to either a segment limit or access rights violation. If a stack segment limit is violated, a stack segment overrun exception (12) occurs.
10. For segment load operations, the CPL, RPL and DPL must agree with privilege rules to avoid an exception. The segment must be present to avoid a not-present exception (11). If the SS register is the destination and a segment not-present violation occurs, a stack exception (12) occurs.
11. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert $\overline{\text{LOCK}}$ to maintain descriptor integrity in multiprocessor systems.
12. JMP, CALL, INT, RET, IRET instructions referring to another code segment will cause a general protection exception (13) if any privilege rule is violated.
13. A general protection exception (13) occurs if CPL $\neq$ 0.
14. A general protection exception (13) occurs if CPL > IOPL.
15. The IF field of the flag word is not updated if CPL > IOPL. The IOPL field is updated only if CPL = 0.
16. Any violation of privilege rules as applied to the selector operand does not cause a protection exception; rather, the instruction does not return a result and the zero flag is cleared.
17. If the starting address of the memory operand violates a segment limit, or an invalid access is attempted, a general protection exception (13) will occur before the ESC instruction is executed. A stack segment overrun exception (12) will occur if the stack limit is violated by the operand's starting address. If a segment limit is violated during an attempted data transfer then a processor extension segment overrun exception (9) occurs.
18. The destination of an INT, JMP, CALL, RET or IRET instruction must be in the defined limit of a code segment or a general protection exception (13) will occur.

80C286 Instruction Set Summary

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
DATA TRANSFER								
MOV = Move								
Register to Register/Memory	1000100w	mod reg r/m			2, 3 (Note 59)	2, 3 (Note 59)	2	9
Register/Memory to Register	1000101w	mod reg r/m			2, 5 (Note 59)	2, 5 (Note 59)	2	9
Immediate to Register/Memory	1100011w	mod 000 r/m	data	data if w = 1	2, 3 (Note 59)	2, 3 (Note 59)	2	9
Immediate to Register	1011w reg	data	data if w = 1		2	2		
Memory to Accumulator	1010000w	addr-low	addr-high		5	5	2	9

80C286 Instruction Set Summary **(Continued)**

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
Accumulator to Memory	1010001w	addr-low	addr-high		3	3	2	9
Register/Memory to Segment Register	10001110	mod 0 reg r/m			2, 5 (Note 59)	17, 19 (Note 59)	2	9, 10, 11
Segment Register to Register/Memory	10001100	mod 0 reg r/m			2, 3 (Note 59)	2, 3 (Note 59)	2	9
PUSH = Push								
Memory	11111111	mod 110 r/m			5 (Note 59)	5 (Note 59)	2	9
Register	01010 reg				3	3	2	9
Segment Register	000 reg 110				3	3	2	9
Immediate	011010s0	data	data if s = 0		3	3	2	9
PUSHA = Push All	01100000				17	17	2	9
POP = Pop								
Memory	10001111	mod 000 r/m			5 (Note 59)	5 (Note 59)	2	9
Register	01011 reg				5	5	2	9
Segment Register	000 reg 111	(reg ≠ 01)			5	20	2	9, 10, 11
POPA = Pop All	01100001				19	19	2	9
XCHG = Exchange								
Register/Memory with Register	1000011w	mod reg r/m			3, 5 (Note 59)	3, 5 (Note 59)	2, 7	7, 9
Register with Accumulator	10010 reg				3	3		
IN = Input From								
Fixed Port	1110010w	port			5	5		14
Variable Port	1110110w				5	5		14
OUT = Output To								
Fixed Port	1110011w	port			3	3		14
Variable Port	1110111w				3	3		14
XLAT = Translate Byte to AL	11010111				5	5		9

80C286 Instruction Set Summary **(Continued)**

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
Immediate from Register/Memory	100000sw	mod 101 r/m	data	data if sw = 01	3, 7 (Note 59)	3, 7 (Note 59)	2	9
Immediate from Accumulator	0010110w	data	data if w = 1		3	3		
SBB = Subtract with Borrow								
Reg/Memory and Register to Either	000110dw	mod reg r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9
Immediate from Register/Memory	100000sw	mod 011 r/m	data	data if sw = 01	3, 7 (Note 59)	3, 7 (Note 59)	2	9
Immediate from Accumulator	0001110w	data	data if w = 1		3	3		
DEC = Decrement								
Register/Memory	1111111w	mod 001 r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9
Register	01001 reg				2	2		
CMP = Compare								
Register/Memory with Register	0011101w	mod reg r/m			2, 6 (Note 59)	2, 6 (Note 59)	2	9
Register with Register/Memory	0011100w	mod reg r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9
Immediate with Register/Memory	100000sw	mod 111 r/m	data	data if sw = 01	3, 6 (Note 59)	3, 6 (Note 59)	2	9
Immediate with Accumulator	0011110w	data	data if w = 1		3	3		
NEG = Change Sign	1111011w	mod 011 r/m			2	7 (Note 59)	2	7
AAA = ASCII Adjust for Add	00110111				3	3		
DAA = Decimal Adjust for Add	00100111				3	3		
AAS = ASCII Adjust for Subtract	00111111				3	3		
DAS = Decimal Adlust for Subtract	00101111				3	3		
MUL = Multiply (Unsigned)	1111011w	mod 100 r/m						

80C286 Instruction Set Summary **(Continued)**

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
Register - Byte					13	13		
Register - Word					21	21		
Memory - Byte					16 (Note 59)	16 (Note 59)	2	9
Memory - Word					24 (Note 59)	24 (Note 59)	2	9
IMUL = Integer Multiply (Signed)	1111011w	mod 101 r/m						
Register - Byte					13	13		
Register - Word					21	21		
Memory - Byte					16 (Note 59)	16 (Note 59)	2	9
Memory - Word					24 (Note 59)	24 (Note 59)	2	9
IMUL = Interger Immediate Multiply (Signed)	011010s1	mod reg r/m	data	data if s = 0	21, 24 (Note 59)	21, 24 (Note 59)	2	9
DIV = Divide (Unsigned)	1111011w	mod 110 r/m						
Register - Byte					14	14	6	6
Register - Word					22	22	6	6
Memory - Byte					17 (Note 59)	17 (Note 59)	2, 6	6, 9
Memory - Word					25 (Note 59)	25 (Note 59)	2, 6	6, 9
IDIV = Integer Divide (Signed)	1111011w	mod 111 r/m						
Register - Byte					17	17	6	6
Register - Word					25	25	6	6
Memory - Byte					20 (Note 59)	20 (Note 59)	2, 6	6, 9
Memory - Word					28 (Note 59)	28 (Note 59)	2, 6	6, 9
AAM = ASCII Adjust for Multiply	11010100	00001010			16	16		

80C286 Instruction Set Summary **(Continued)**

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
Immediate Data and Register/Memory	1111011w	mod 000 r/m	data	data if w = 1	3, 6 (Note 59)	3, 6 (Note 59)	2	9
Immediate Data and Accumulator	1010100w	data	data if w = 1		3	3		
OR = Or								
Reg/Memory and Register to Either	000010dw	mod reg r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9
Immediate to Register/Memory	1000000w	mod 001 r/m	data	data if w = 1	3, 7 (Note 59)	3, 7 (Note 59)	2	9
Immediate to Accumulator	0000110w	data	data if w = 1		3	3		
XOR = Exclusive or								
Reg/Memory and Register to Either	001100dw	mod reg r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9
Immediate to Register/Memory	1000000w	mod reg r/m	data	data if w = 1	3, 7 (Note 59)	3, 7 (Note 59)	2	9
Immediate to Accumulator	0011010w	data	data if w = 1		3	3		
NOT = Invert Register/Memory	1111011w	mod 010 r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9
STRING MANIPULATION								
MOVS = Move Byte/Word	1010010w				5	5	2	9
CMPS = Compare Byte/Word	1010011w				8	8	2	9
SCAS = Scan Byte/Word	1010111w				7	7	2	9
LODS = Load Byte/Word to AL/AX	1010110w				5	5	2	9
STOS = Store Byte/Word from AL/A	1010101w				3	3	2	9
INS = Input Byte/Word from DX Port	0110110w				5	5	2	9, 14
OUTS = Output Byte/Word to DX Port	0110111w				5	5	2	9, 14