

## High Performance Microprocessor with Memory Management and Protection

March 1997

### Features

- **Compatible with NMOS 80286**
- **Wide Range of Clock Rates**
  - DC to 25MHz (80C286-25)
  - DC to 20MHz (80C286-20)
  - DC to 16MHz (80C286-16)
  - DC to 12.5MHz (80C286-12)
  - DC to 10MHz (80C286-10)
- **Static CMOS Design for Low Power Operation**
  - ICCSB = 5mA Maximum
  - ICCOP = 185mA Maximum (80C286-10)  
220mA Maximum (80C286-12)  
260mA Maximum (80C286-16)  
310mA Maximum (80C286-20)  
410mA Maximum (80C286-25)
- **High Performance Processor (Up to 19 Times the 8086 Throughput)**
- **Large Address Space**
- **16 Megabytes Physical/1 Gigabyte Virtual per Task**
- **Integrated Memory Management, Four-Level Memory Protection and Support for Virtual Memory and Operating Systems**
- **Two 80C86 Upward Compatible Operating Modes**
  - 80C286 Real Address Mode
  - PVAM
- **Compatible with 80287 Numeric Data Co-Processor**
- **High Bandwidth Bus Interface (25 Megabyte/Sec)**
- **Available In**
  - 68 Pin PGA (Commercial, Industrial, and Military)
  - 68 Pin PLCC (Commercial and Industrial)

### Description

The Harris 80C286 is a static CMOS version of the NMOS 80286 microprocessor. The 80C286 is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. The 80C286 has built-in memory protection that supports operating system and task isolation as well as program and data privacy within tasks. A 25MHz 80C286 provides up to nineteen times the throughput of a standard 5MHz 8086. The 80C286 includes memory management capabilities that map  $2^{30}$  (one gigabyte) of virtual address space per task into  $2^{24}$  bytes (16 megabytes) of physical memory.

The 80C286 is upwardly compatible with 80C86 and 80C88 software (the 80C286 instruction set is a superset of the 80C86/80C88 instruction set). Using the 80C286 real address mode, the 80C286 is object code compatible with existing 80C86 and 80C88 software. In protected virtual address mode, the 80C286 is source code compatible with 80C86 and 80C88 software but may require upgrading to use virtual address as supported by the 80C286's integrated memory management and protection mechanism. Both modes operate at full 80C286 performance and execute a superset of the 80C86 and 80C88 instructions.

The 80C286 provides special operations to support the efficient implementation and execution of operating systems. For example, one instruction can end execution of one task, save its state, switch to a new task, load its state, and start execution of the new task. The 80C286 also supports virtual memory systems by providing a segment-not-present exception and restartable instructions.

### Ordering Information

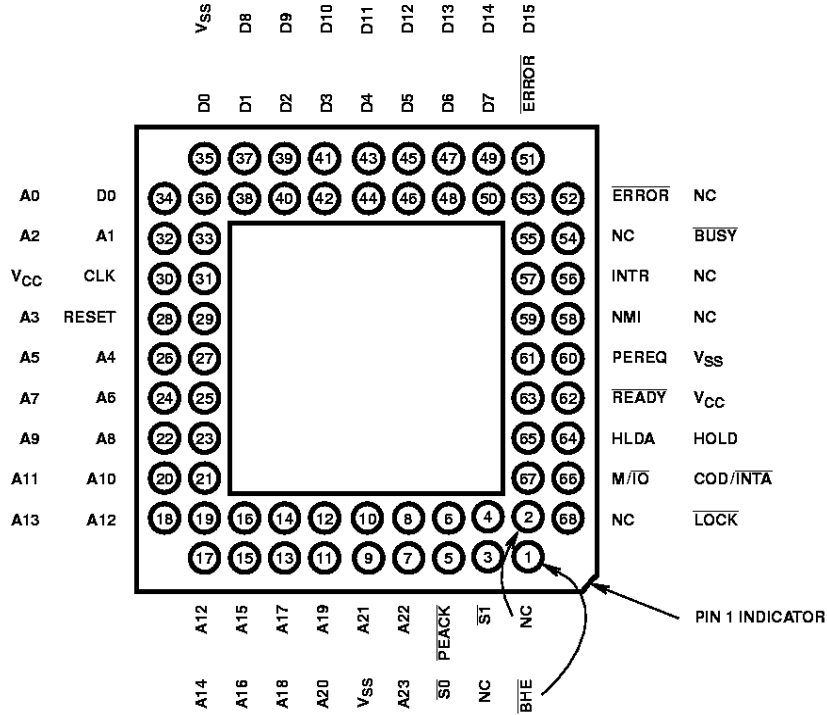
PACKAGE	TEMP. RANGE	10MHz	12.5MHz	16MHz	20MHz	25MHz	PKG. NO.
PGA	0°C to +70°C	-	CG80C286-12	CG80C286-16	CG80C286-20	-	G68.B
	-40°C to +85°C	IG80C286-10	IG80C286-12	-	-	-	G68.B
	-55°C to +125°C	5962-9067801MXC	5962-9067802MXC	-	-	-	G68.B
PLCC	0°C to +70°C	-	CS80C286-12	CS80C286-16	CS80C286-20	CS80C286-25	N68.95
	-40°C to +85°C	IS80C286-10	IS80C286-12	IS80C286-16	IS80C286-20	-	N68.95

# 80C286

## Pinouts

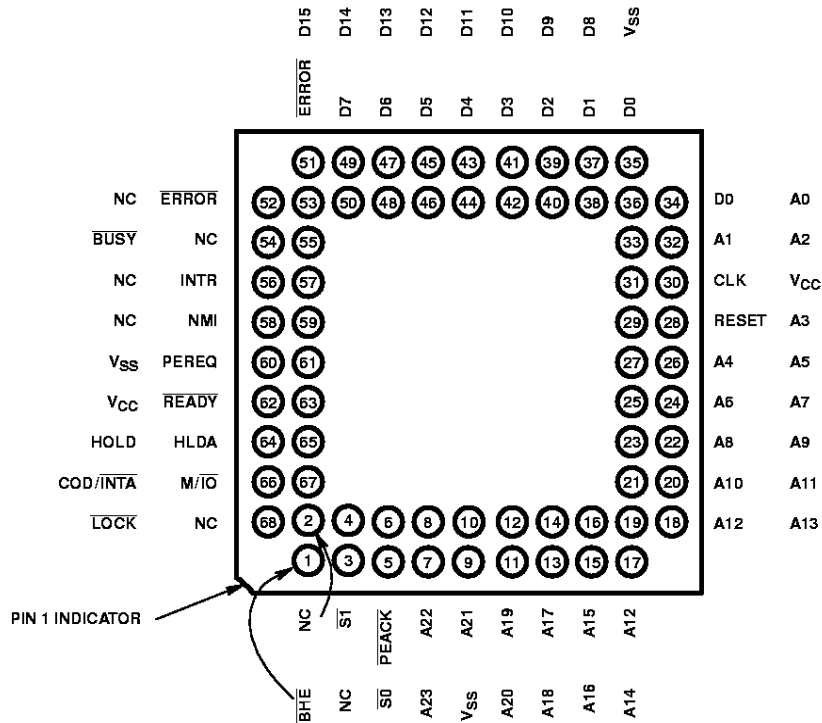
### 68 LEAD PGA

Component Pad View - As viewed from underside of the component when mounted on the board.



### 68 LEAD PGA

P.C. Board View - As viewed from the component side of the P.C. board.

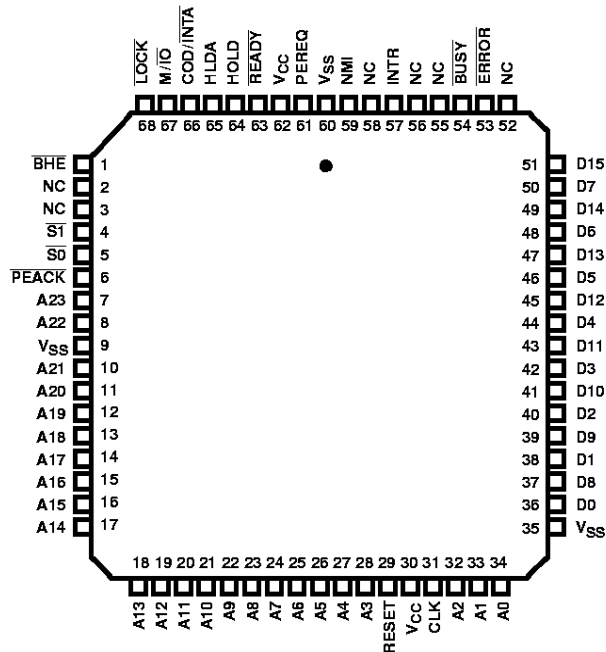


# 80C286

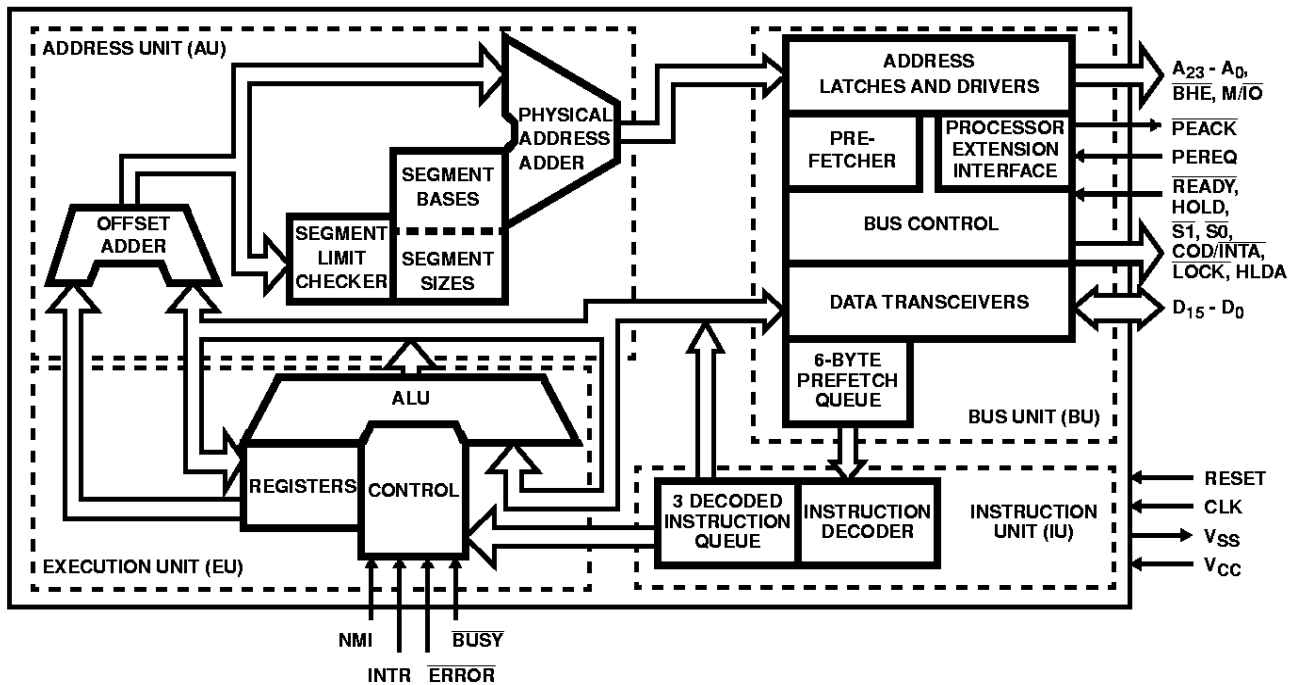
## Pinouts (Continued)

### 68 LEAD PLCC

P.C. Board View - As viewed from the component side of the P.C. board.



## Functional Diagram



## 80C286

**Pin Descriptions** The following pin function descriptions are for the 80C286 microprocessor.

SYMBOL	PIN NUMBER	TYPE	DESCRIPTION																																																																																					
CLK	31	I	SYSTEM CLOCK: provides the fundamental timing for the 80C286 system. It is divided by two inside the 80C286 to generate the processor clock. The internal divide-by-two circuitry can be synchronized to an external clock generator by a LOW to HIGH transition on the RESET input.																																																																																					
D <sub>15</sub> - D <sub>0</sub>	36 - 51	I/O	DATA BUS: inputs data during memory, I/O, and interrupt acknowledge read cycles; outputs data during memory and I/O write cycles. The data bus is active HIGH and is held at high impedance to the last valid logic level during bus hold acknowledge.																																																																																					
A <sub>23</sub> - A <sub>0</sub>	7 - 8 10 - 28 32 - 43	O	ADDRESS BUS: outputs physical memory and I/O port addresses. A <sub>23</sub> - A <sub>16</sub> are LOW during I/O transfers. A <sub>0</sub> is LOW when data is to be transferred on pins D <sub>7</sub> - D <sub>0</sub> (see table below). The address bus is active High and floats to three-state off during bus hold acknowledge.																																																																																					
$\overline{\text{BHE}}$	1	O	BUS HIGH ENABLE: indicates transfer of data on the upper byte of the data bus, D <sub>15</sub> - D <sub>8</sub> . Eight-bit oriented devices assigned to the upper byte of the data bus would normally use BHE to condition chip select functions. BHE is active LOW and floats to three-state OFF during bus hold acknowledge.  <b><math>\overline{\text{BHE}}</math> AND A<sub>0</sub> ENCODINGS</b>  <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <thead> <tr> <th style="width: 25%;">BHE VALUE</th> <th style="width: 25%;">A<sub>0</sub> VALUE</th> <th style="width: 50%;">FUNCTION</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>Word transfer</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>Byte transfer on upper half of data bus (D<sub>15</sub> - D<sub>8</sub>)</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>Byte transfer on lower half of data bus (D<sub>7</sub> - D<sub>0</sub>)</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>Reserved</td> </tr> </tbody> </table>	BHE VALUE	A <sub>0</sub> VALUE	FUNCTION	0	0	Word transfer	0	1	Byte transfer on upper half of data bus (D <sub>15</sub> - D <sub>8</sub> )	1	0	Byte transfer on lower half of data bus (D <sub>7</sub> - D <sub>0</sub> )	1	1	Reserved																																																																						
BHE VALUE	A <sub>0</sub> VALUE	FUNCTION																																																																																						
0	0	Word transfer																																																																																						
0	1	Byte transfer on upper half of data bus (D <sub>15</sub> - D <sub>8</sub> )																																																																																						
1	0	Byte transfer on lower half of data bus (D <sub>7</sub> - D <sub>0</sub> )																																																																																						
1	1	Reserved																																																																																						
$\overline{\text{S1}}$ , $\overline{\text{S0}}$	4, 5	O	BUS CYCLE STATUS: indicates initiation of a bus cycle and along with M/ $\overline{\text{IO}}$ and COD/ $\overline{\text{INTA}}$ , defines the type of bus cycle. The bus is in a T <sub>S</sub> state whenever one or both are LOW. $\overline{\text{S1}}$ and $\overline{\text{S0}}$ are active LOW and are held at a high impedance logic one during bus hold acknowledge.  <b>80C286 BUS CYCLE STATUS DEFINITION</b>  <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">COD/<math>\overline{\text{INTA}}</math></th> <th style="width: 10%;">M/<math>\overline{\text{IO}}</math></th> <th style="width: 10%;"><math>\overline{\text{S1}}</math></th> <th style="width: 10%;"><math>\overline{\text{S0}}</math></th> <th style="width: 60%;">BUS CYCLE INITIATED</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0(Low)</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>Interrupt acknowledge</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>Reserved</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>Reserved</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>None; not a status cycle</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>If A<sub>1</sub> = 1 then halt; else shutdown</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>Memory data read</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>Memory data write</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>None; not a status cycle</td> </tr> <tr> <td style="text-align: center;">1(High)</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>Reserved</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>I/O read</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>I/O write</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>None; not a status cycle</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td>Reserved</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td>Memory instruction read</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td>Reserved</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td>None; not a status cycle</td> </tr> </tbody> </table>	COD/ $\overline{\text{INTA}}$	M/ $\overline{\text{IO}}$	$\overline{\text{S1}}$	$\overline{\text{S0}}$	BUS CYCLE INITIATED	0(Low)	0	0	0	Interrupt acknowledge	0	0	0	1	Reserved	0	0	1	0	Reserved	0	0	1	1	None; not a status cycle	0	1	0	0	If A <sub>1</sub> = 1 then halt; else shutdown	0	1	0	1	Memory data read	0	1	1	0	Memory data write	0	1	1	1	None; not a status cycle	1(High)	0	0	0	Reserved	1	0	0	1	I/O read	1	0	1	0	I/O write	1	0	1	1	None; not a status cycle	1	1	0	0	Reserved	1	1	0	1	Memory instruction read	1	1	1	0	Reserved	1	1	1	1	None; not a status cycle
COD/ $\overline{\text{INTA}}$	M/ $\overline{\text{IO}}$	$\overline{\text{S1}}$	$\overline{\text{S0}}$	BUS CYCLE INITIATED																																																																																				
0(Low)	0	0	0	Interrupt acknowledge																																																																																				
0	0	0	1	Reserved																																																																																				
0	0	1	0	Reserved																																																																																				
0	0	1	1	None; not a status cycle																																																																																				
0	1	0	0	If A <sub>1</sub> = 1 then halt; else shutdown																																																																																				
0	1	0	1	Memory data read																																																																																				
0	1	1	0	Memory data write																																																																																				
0	1	1	1	None; not a status cycle																																																																																				
1(High)	0	0	0	Reserved																																																																																				
1	0	0	1	I/O read																																																																																				
1	0	1	0	I/O write																																																																																				
1	0	1	1	None; not a status cycle																																																																																				
1	1	0	0	Reserved																																																																																				
1	1	0	1	Memory instruction read																																																																																				
1	1	1	0	Reserved																																																																																				
1	1	1	1	None; not a status cycle																																																																																				

## 80C286

**Pin Descriptions** The following pin function descriptions are for the 80C286 microprocessor. (Continued)

SYMBOL	PIN NUMBER	TYPE	DESCRIPTION
$M/\overline{IO}$	67	O	MEMORY I/O SELECT: distinguishes memory access from I/O access. If HIGH during $T_S$ , a memory cycle or a halt/shutdown cycle is in progress. If LOW, an I/O cycle or an interrupt acknowledge cycle is in progress. $M/\overline{IO}$ is held at high impedance to the last valid logic state during bus hold acknowledge.
$COD/\overline{INTA}$	66	O	CODE/INTERRUPT ACKNOWLEDGE: distinguishes instruction fetch cycles from memory data read cycles. Also distinguishes interrupt acknowledge cycles from I/O cycles. $COD/\overline{INTA}$ is held at high impedance to the last valid logic state during bus hold acknowledge. Its timing is the same as $M/\overline{IO}$ .
$\overline{LOCK}$	68	O	BUS LOCK: indicates that other system bus masters are not to gain control of the system bus for the current and following bus cycles. The $\overline{LOCK}$ signal may be activated explicitly by the "LOCK" instruction prefix or automatically by 80C286 hardware during memory XCHG instructions, interrupt acknowledge, or descriptor table access. $\overline{LOCK}$ is active LOW and is held at a high impedance logic one during bus hold acknowledge.
READY	63	I	BUS READY: terminates a bus cycle. Bus cycles are extended without limit until terminated by READY LOW. READY is an active LOW synchronous input requiring setup and hold times relative to the system clock be met for correct operation. READY is ignored during bus hold acknowledge. (See Note 1)
HOLD HLDA	64 65	I O	BUS HOLD REQUEST AND HOLD ACKNOWLEDGE: control ownership of the 80C286 local bus. The HOLD input allows another local bus master to request control of the local bus. When control is granted, the 80C286 will float its bus drivers and then activate HLDA, thus entering the bus hold acknowledge condition. The local bus will remain granted to the requesting master until HOLD becomes inactive which results in the 80C286 deactivating HLDA and regaining control of the local bus. This terminates the bus hold acknowledge condition. HOLD may be asynchronous to the system clock. These signals are active HIGH. Note that HLDA never floats.
INTR	57	I	INTERRUPT REQUEST: requires the 80C286 to suspend its current program execution and service a pending external request. Interrupt requests are masked whenever the interrupt enable bit in the flag word is cleared. When the 80C286 responds to an interrupt request, it performs two interrupt acknowledge bus cycles to read an 8-bit interrupt vector that identifies the source of the interrupt. To ensure program interruption, INTR must remain active until an interrupt acknowledge bus cycle is initiated. INTR is sampled at the beginning of each processor cycle and must be active HIGH at least two processor cycles before the current instruction ends in order to interrupt before the next instruction. INTR is level sensitive, active HIGH, and may be asynchronous to the system clock.
NMI	59	I	NON-MASKABLE INTERRUPT REQUEST: interrupts the 80C286 with an internally supplied vector value of two. No interrupt acknowledge cycles are performed. The interrupt enable bit in the 80C286 flag word does not affect this input. The NMI input is active HIGH, may be asynchronous to the system clock, and is edge triggered after internal synchronization. For proper recognition, the input must have been previously LOW for at least four system clock cycles and remain HIGH for at least four system clock cycles.
PEREQ PEACK	61 6	I O	PROCESSOR EXTENSION OPERAND REQUEST AND ACKNOWLEDGE: extend the memory management and protection capabilities of the 80C286 to processor extensions. The PEREQ input requests the 80C286 to perform a data operand transfer for a processor extension. The PEACK output signals the processor extension when the requested operand is being transferred. PEREQ is active HIGH. PEACK is active LOW and is held at a high impedance logic one during bus hold acknowledge. PEREQ may be asynchronous to the system clock.
$\overline{BUSY}$ ERROR	54 53	I I	PROCESSOR EXTENSION BUSY AND ERROR: indicates the operating condition of a processor extension to the 80C286. An active $\overline{BUSY}$ input stops 80C286 program execution on WAIT and some ESC instructions until $\overline{BUSY}$ becomes inactive (HIGH). The 80C286 may be interrupted while waiting for $\overline{BUSY}$ to become inactive. An active ERROR input causes the 80C286 to perform a processor extension interrupt when executing WAIT or some ESC instructions. These inputs are active LOW and may be asynchronous to the system clock.

### Pin Descriptions

The following pin function descriptions are for the 80C286 microprocessor. (Continued)

SYMBOL	PIN NUMBER	TYPE	DESCRIPTION								
RESET	29	I	<p>SYSTEM RESET: clears the internal logic of the 80C286 and is active HIGH. The 80C286 may be reinitialize at any time with a LOW to HIGH transition on RESET which remains active for more than 16 system clock cycles. During RESET active, the output pins of the 80C286 enter the state shown below.</p> <p style="text-align: center;"><b>80C286 PIN STATE DURING RESET</b></p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>PIN VALUE</th> <th>PIN NAMES</th> </tr> </thead> <tbody> <tr> <td>1 (HIGH)</td> <td>S<sub>0</sub>, S<sub>1</sub>, PEACK, A<sub>23</sub> - A<sub>0</sub>, BHE, LOCK</td> </tr> <tr> <td>0 (LOW)</td> <td>M/<math>\overline{IO}</math>, COD/<math>\overline{INTA}</math>, HLDA (Note 2)</td> </tr> <tr> <td>HIGH IMPEDANCE</td> <td>D15 - D0</td> </tr> </tbody> </table> <p>Operation of the 80C286 begins after a HIGH to LOW transition on RESET. The HIGH to LOW transition of RESET must be synchronous to the system clock. Approximately 50 system clock cycles are required by the 80C286 for internal initializations before the first bus cycle to fetch code from the power-on execution address is performed. A LOW to HIGH transition of RESET synchronous to the system clock will end a processor cycle at the second HIGH to LOW transition of the system clock. The LOW to HIGH transition of RESET may be asynchronous to the system clock; however, in this case it cannot be predetermined which phase of the processor clock will occur during the next system clock period. Synchronous LOW to HIGH transitions of RESET are required only for systems where the processor clock must be phase synchronous to another clock.</p>	PIN VALUE	PIN NAMES	1 (HIGH)	S <sub>0</sub> , S <sub>1</sub> , PEACK, A <sub>23</sub> - A <sub>0</sub> , BHE, LOCK	0 (LOW)	M/ $\overline{IO}$ , COD/ $\overline{INTA}$ , HLDA (Note 2)	HIGH IMPEDANCE	D15 - D0
PIN VALUE	PIN NAMES										
1 (HIGH)	S <sub>0</sub> , S <sub>1</sub> , PEACK, A <sub>23</sub> - A <sub>0</sub> , BHE, LOCK										
0 (LOW)	M/ $\overline{IO}$ , COD/ $\overline{INTA}$ , HLDA (Note 2)										
HIGH IMPEDANCE	D15 - D0										
V <sub>SS</sub>	9, 35, 60	I	SYSTEM GROUND: are the ground pins (all must be connected to system ground).								
V <sub>CC</sub>	30, 62	I	SYSTEM POWER: +5V power supply pins. A 0.1 $\mu$ F capacitor between pins 60 and 62 is recommended.								

#### NOTES:

1. READY is an open-collector signal and should be pulled inactive with an appropriate resistor (620 $\Omega$  at 10MHz and 12.5MHz, 470 $\Omega$  at 16MHz, 390 $\Omega$  at 20MHz, 270 $\Omega$  at 25MHz).
2. HLDA is only Low if HOLD is inactive (Low).
3. All unused inputs should be pulled to their inactive state with pull up/down resistors.

## Functional Description

### Introduction

The Harris 80C286 microprocessor is a static CMOS version of the NMOS 80286 microprocessor. The 80C286 is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. Depending on the application, the 80C286's performance is up to nineteen times faster than the standard 5MHz 8086's, while providing complete upward software compatibility with Harris 80C86 and 80C88 CPU family.

The 80C286 operates in two modes: 80C286 real address mode and protected virtual address mode. Both modes execute a superset of the 80C86 and 80C88 instruction set.

In 80C286 real address mode programs use real addresses with up to one megabyte of address space. Programs use virtual addresses in protected virtual address mode, also called protected mode. In protected mode, the 80C286 CPU automatically maps 1 gigabyte of virtual addresses per task into a 16 megabyte real address space. This mode also provides memory protection to isolate the operating system and ensure privacy of each tasks' programs and data. Both modes provide the same base instruction set, registers and addressing modes.

The Functional Description describes the following: Static operation, the base 80C286 architecture common to both modes, 80C286 real address mode, and finally, protected mode.

### Static Operation

The 80C286 is comprised of completely static circuitry. Internal registers, counters, and latches are static and require no refresh as with dynamic circuit design. This eliminates the minimum operating frequency restriction typically placed on microprocessors. The CMOS 80C286 can operate from DC to the specified upper frequency limit. The clock to the processor may be stopped at any point (either phase one or phase two of the processor clock cycle) and held there indefinitely. There is, however, a significant decrease in power requirement if the clock is stopped in phase two of the processor clock cycle. Details on the clock relationships will be discussed in the Bus Operation section. The ability to stop the clock to the processor is especially useful for system debug or power critical applications.

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.

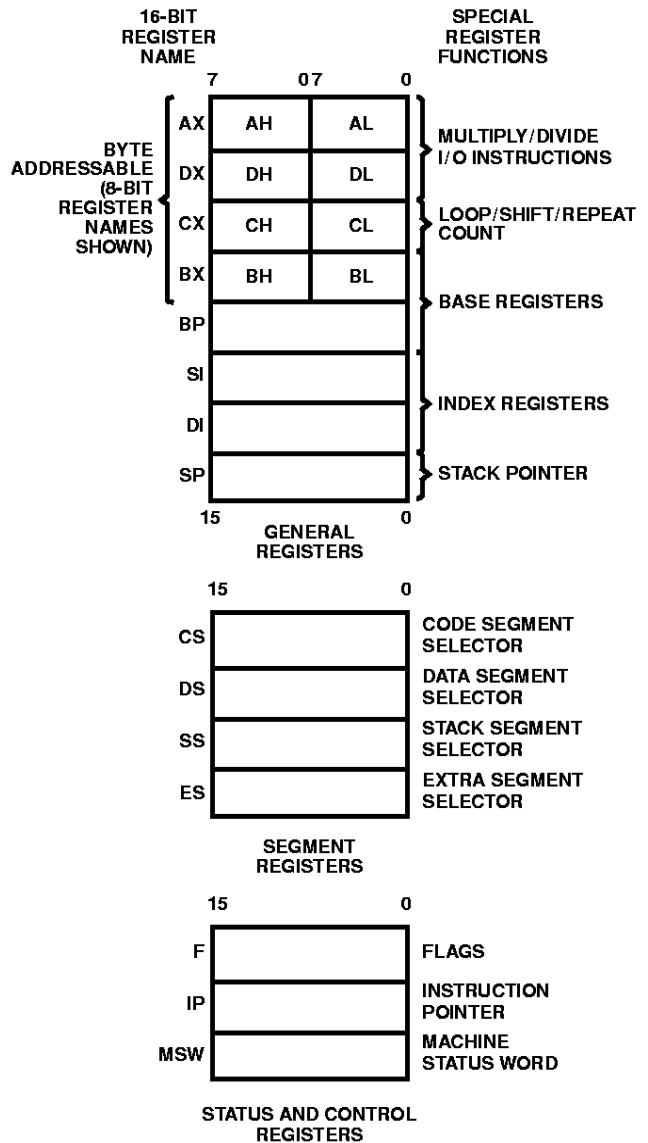


FIGURE 1. REGISTER SET

#### 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.

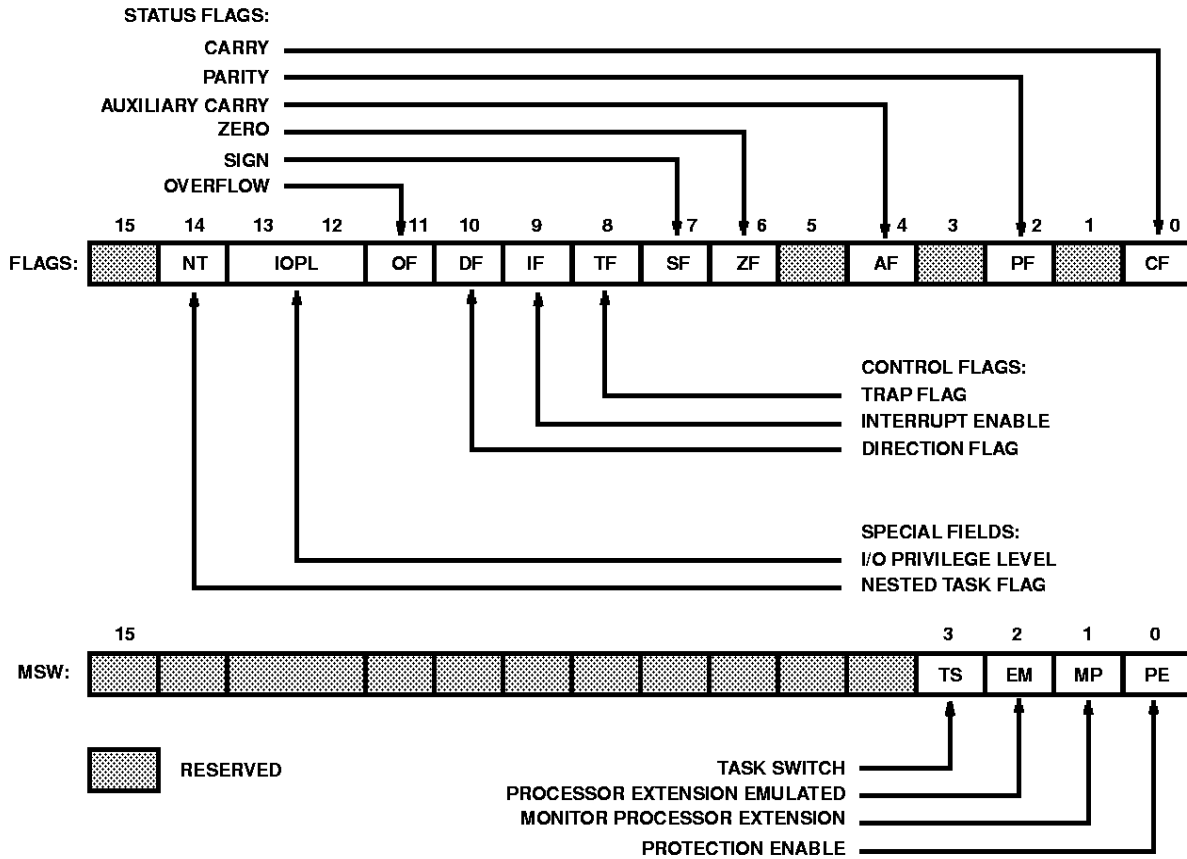


FIGURE 2. STATUS AND CONTROL REGISTER BIT FUNCTIONS

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.



## Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, string manipulation, shift/rotate/logical, high level, processor control and control transfer instructions. These categories are summarized in Table 2.

An 80C286 instruction can reference zero, one, or two operands; where an operand may reside in a register, in the instruction itself, or in memory. Zero-operand instructions (e.g. NOP and HLT) are usually one byte long. One-operand instructions (e.g. INC and DEC) are usually two bytes long but some are encoded in only one byte. One-operand instructions may reference a register or memory location. Two-operand instructions permit the following six types of instruction operations:

- Register to Register
- Memory to Memory
- Memory to Register
- Register to Memory
- Immediate to Register
- Immediate to Memory

Two-operand instructions (e.g. MOV and ADD) are usually three to six bytes long. Memory to memory operations are provided by a special class of string instructions requiring one to three bytes. For detailed instruction formats and encodings refer to the instruction set summary at the end of this document.

**TABLE 2A. DATA TRANSFER INSTRUCTIONS**

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack

**TABLE 2B. ARITHMETIC INSTRUCTIONS**

ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

**TABLE 2C. STRING INSTRUCTIONS**

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero

TABLE 2D. SHIFT/ROTATE LOGICAL INSTRUCTIONS

LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

TABLE 2E. HIGH LEVEL INSTRUCTIONS

ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

TABLE 2F. PROCESSOR CONTROL INSTRUCTIONS

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
EXECUTION ENVIRONMENT CONTROL	
LMSW	Load machine status word
SMSW	Store machine status word

TABLE 2G. PROGRAM TRANSFER INSTRUCTIONS

CONDITIONAL TRANSFERS		UNCONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal	CALL	Call procedure
JAE/JNB	Jump if above or equal/not below	RET	Return from procedure
JB/JNAE	Jump if below/not above nor equal	JMP	Jump
JBE/JNA	Jump if below or equal/not above		
JC	Jump if carry	ITERATION CONTROLS	
JE/JZ	Jump if equal/zero	LOOP	Loop
JG/JNLE	Jump if greater/not less nor equal	LOOPE/LOOPZ	Loop if equal/zero
JGE/JNL	Jump if greater or equal/not less	LOOPNE/LOOPNZ	Loop if not equal/not zero
JL/JNGE	Jump if less/not greater nor equal	JCXZ	Jump if register CX = 0
JLE/JNG	Jump if less or equal/not greater		
JNC	Jump if not carry	INTERRUPTS	
JNE/JNZ	Jump if not equal/not zero	INT	Interrupt
JNO	Jump if not overflow		
JNP/JPO	Jump if not parity/parity odd	INTO	Interrupt if overflow
JNS	Jump if not sign	IRET	Interrupt return
JO	Jump if overflow		
JP/JPE	Jump if parity/parity even		
JS	Jump if sign		

**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.

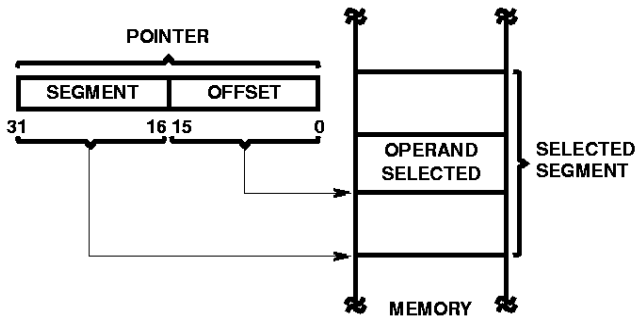


FIGURE 3. TWO COMPONENT ADDRESS

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)

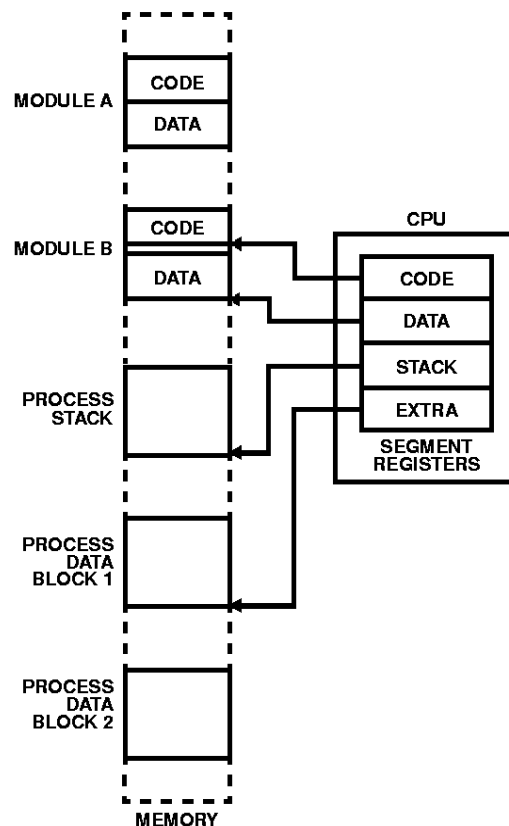


FIGURE 4. SEGMENTED MEMORY HELPS STRUCTURE SOFTWARE

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.

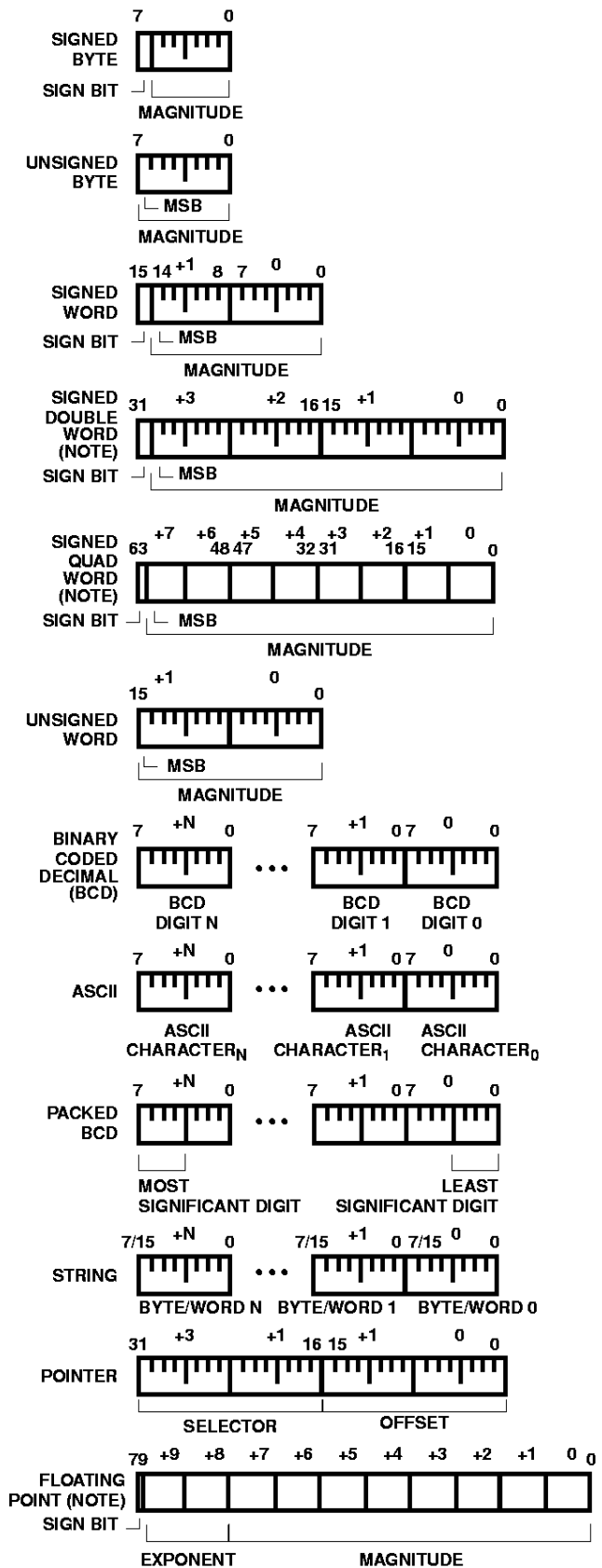


FIGURE 5. 80C286 SUPPORTED DATA TYPES

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<sub>15</sub>-A<sub>8</sub> 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.

### Single Step Interrupt

The 80C286 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single step interrupt and is controlled by the single step flag bit (TF) in the flag word. Once this bit is set, an internal single step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single stepped.

### Interrupt Priorities

When simultaneous interrupt requests occur, they are processed in a fixed order as shown in Table 5. Interrupt processing involves saving the flags, return address, and setting CS:IP to point at the first instruction of the interrupt handler. If another enabled interrupt should occur, it is processed before the next instruction of the current interrupt handler is executed. The last interrupt processed is therefore the first one serviced.

TABLE 5. INTERRUPT PROCESSING ORDER

ORDER	INTERRUPT
1	Instruction Exception
2	Single Step
3	NMI
4	Processor Extension Segment Overrun
5	INTR
6	INT Instruction

### Initialization and Processor Reset

Processor initialization or start up is accomplished by driving the RESET input pin HIGH. RESET forces the 80C286 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active. After RESET becomes inactive, and an internal processing interval elapses, the 80C286 begins execution in real address mode with the instruction at physical location FFFFF0(H). RESET also sets some registers to predefined values as shown in Table 6.

TABLE 6. 80C286 INITIAL REGISTER STATE AFTER RESET

Flag Word	0002(H)
Machine Status Word	FFF0(H)
Instruction Pointer	FFF0(H)
Code Segment	F000(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)

HOLD must not be active during the time from the leading edge of the initial RESET to 34 CLKs after the trailing edge of the initial RESET of an 80C286 system.

### Machine Status Word Description

The machine status word (MSW) records when a task switch takes place and controls the operating mode of the 80C286. It is a 16-bit register of which the lower four bits are used. One bit places the CPU into protected mode, while the other three bits, as shown in Table 7, control the processor extension interface. After RESET, this register contains FFF0(H) which places the 80C286 in 80C286 real address mode.

TABLE 7. MSW BIT FUNCTIONS

BIT POSITION	NAME	FUNCTION
0	PE	Protected mode enable places the 80C286 into protected mode and cannot be cleared except by RESET.
1	MP	Monitor processor extension allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	Emulate processor extension causes a processor extension not present exception (number 7) on ESC instructions to allow emulating a processor extension.
3	TS	Task switched indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task.

The LMSW and SMSW instructions can load and store the MSW in real address mode. The recommended use of TS, EM, and MP is shown in Table 8.

### Halt

The HLT instruction stops program execution and prevents the CPU from using the local bus until restarted. Either NMI, INTR with IF = 1, or RESET will force the 80C286 out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

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<sub>0</sub> through A<sub>19</sub> and BHE. A<sub>20</sub> through A<sub>23</sub> 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<sub>0</sub> through A<sub>19</sub> and BHE. Address bits A<sub>20</sub>-A<sub>23</sub> may not always be zero in real mode. A<sub>20</sub>-A<sub>23</sub> 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.

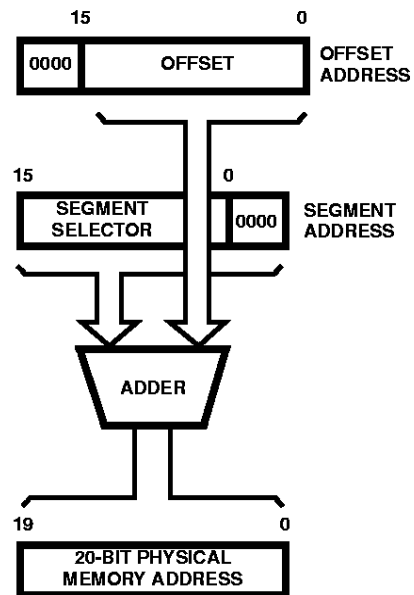
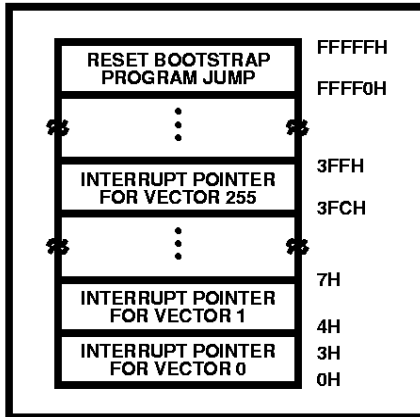


FIGURE 6. 80C286 REAL ADDRESS MODE ADDRESS CALCULATION

## Reserved Memory Locations

The 80C286 reserves two fixed areas of memory in real address mode (see Figure 7); system initialization area and interrupt table area. Locations from addresses FFFF0(H) through FFFFF(H) are reserved for system initialization. Initial execution begins at location FFFF0(H). Locations 00000(H) through 003FF(H) are reserved for interrupt vectors.



INITIAL CS:IP VALUE IS F000:FFF0

FIGURE 7. 80C286 REAL ADDRESS MODE INITIALLY RESERVED MEMORY LOCATIONS

## Interrupts

Table 9 shows the interrupt vectors reserved for exceptions and interrupts which indicate an addressing error. The exceptions leave the CPU in the state existing before attempting to execute the failing instruction (except for PUSH, POP, PUSHA, or POPA). Refer to the next section on protected mode initialization for a discussion on exception 8.

## Protected Mode Initialization

To prepare the 80C286 for protected mode, the LIDT instruction is used to load the 24-bit interrupt table base and 16-bit limit for the protected mode interrupt table. This instruction can also set a base and limit for the interrupt vector table in real address mode. After reset, the interrupt table base is initialized to 00000(H) and its size set to 03FF(H). These values are compatible with 80C86 and 80C88 software. LIDT should only be executed in preparation for protected mode.

## Shutdown

Shutdown occurs when a severe error is detected that prevents further instruction processing by the CPU. Shutdown and halt are externally signalled via a halt bus operation. They can be distinguished by  $A_1$  HIGH for halt and  $A_1$  LOW for shutdown. In real address mode, shutdown can occur under two conditions:

- Exceptions 8 or 13 happen and the IDT limit does not include the interrupt vector.
- A CALL INT or PUSH instruction attempts to wrap around the stack segment when SP is not even.

An NMI input can bring the CPU out of shutdown if the IDT limit is at least 000F(H) and SP is greater than 0005(H), otherwise shutdown can only be exited via the RESET input.

## Protected Virtual Address Mode

The 80C286 executes a fully upward-compatible superset of the 80C86 instruction set in protected virtual address mode (protected mode). Protected mode also provides memory management and protection mechanisms and associated instructions.

The 80C286 enters protected virtual address mode from real address mode by setting the PE (Protection Enable) bit of the machine status word with the Load Machine Status Word (LMSW) instruction. Protected mode offers extended physical and virtual memory address space, memory protection mechanisms, and new operations to support operating systems and virtual memory.

All registers, instructions, and addressing modes described in the 80C286 Base Architecture section of this Functional Description remain the same. Programs for the 80C86, 80C88, and real address mode 80C286 can be run in protected mode; however, embedded constants for segment selectors are different.

## Memory Size

The protected mode 80C286 provides a 1 gigabyte virtual address space per task mapped into a 16 megabyte physical address space defined by the address pins  $A_{23}$ - $A_0$  and  $BHE$ . The virtual address space may be larger than the physical address space since any use of an address that does not map to a physical memory location will cause a restartable exception.

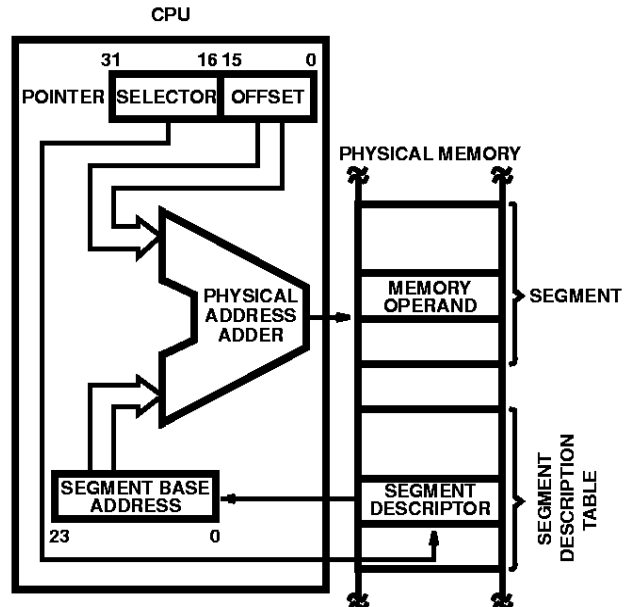


FIGURE 8. PROTECTED MODE MEMORY ADDRESSING

## Memory Addressing

As in real address mode, protected mode uses 32-bit pointers, consisting of 16-bit selector and offset components. The selector, however, specifies an index into a memory resident table rather than the upper 16-bits of a real memory address. The 24-bit base address of the desired segment is obtained



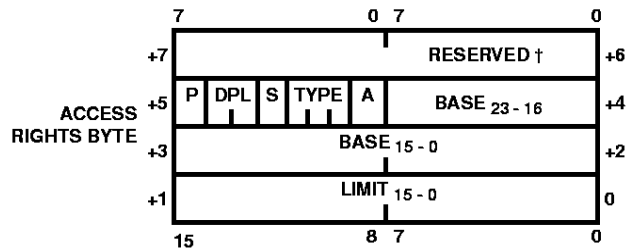
from the tables in memory. The 16-bit offset is added to the segment base address to form the physical address as shown in Figure 8. The tables are automatically referenced by the CPU whenever a segment register is loaded with a selector. All 80C286 instructions which load a segment register will reference the memory based tables without additional software. The memory based tables contain 8 byte values called descriptors.

**Descriptors**

Descriptors define the use of memory. Special types of descriptors also define new functions for transfer of control and task switching. The 80C286 has segment descriptors for code, stack and data segments, and system control descriptors for special system data segments and control transfer operations. Descriptor accesses are performed as locked bus operations to assure descriptor integrity in multi-processor systems.

**Code and Data Segment Descriptors (S = 1)**

Besides segment base addresses, code and data descriptors contain other segment attributes including segment size (1 to 64K bytes), access rights (read only, read/write, execute only, and execute/read), and presence in memory (for virtual memory systems) (See Table 10). Any segment usage violating a segment attribute indicated by the segment descriptor will prevent the memory cycle and cause an exception or interrupt.



† MUST BE SET TO 0 FOR COMPATIBILITY WITH FUTURE UPGRADES

**FIGURE 9. CODE OR DATA SEGMENT DESCRIPTOR**

Code and data (including stack data) are stored in two types of segments: code segments and data segments. Both types are identified and defined by segment descriptors (S = 1). Code segments are identified by the executable (E) bit set to 1 in the descriptor access rights byte. The access rights byte of both code and data segment descriptor types have three fields in common: present (P) bit, Descriptor Privilege Level (DPL), and accessed (A) bit. If P = 0, any attempted use of this segment will cause a not-present exception. DPL specifies the privilege level of the segment descriptor. DPL controls when the descriptor may be used by a task (refer to privilege discussion below). The A bit shows whether the segment has been previously accessed for usage profiling, a necessity for virtual memory systems. The CPU will always set this bit when accessing the descriptor.

**TABLE 10. CODE AND DATA SEGMENT DESCRIPTOR FORMATS - ACCESS RIGHTS BYTE DEFINITION**

BIT POSITION	NAME	FUNCTION	
7	Present (P)	P = 1	Segment is mapped into physical memory.
		P = 0	No mapping to physical memory exists, base and limit are not used.
6 - 5	Descriptor Privilege Level (DPL)		Segment privilege attribute used in privilege tests.
4	Segment Descriptor (S)	S = 1	Code or Data (includes stacks) segment descriptor
		S = 0	System Segment Descriptor or Gate Descriptor
3	Executable (E)	E = 0	Data segment descriptor type is:
2	Expansion Direction (ED)	ED = 0	Expand up segment, offsets must be ≤ limit.
		ED = 1	Expand down segment, offsets must be > limit.
1	Writable (W)	W = 0	Data segment may not be written into.
		W = 1	Data segment may be written into.
3	Executable (E)	E = 1	Code Segment Descriptor type is:
		E = 0	
2	Conforming (C)	C = 1	Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.
1	Readable (R)	R = 0	Code segment may not be read.
		R = 1	Code segment may be read.
0	Accessed (A)	A = 0	Segment has not been accessed.
		A = 1	Segment selector has been loaded into segment register or used by selector test instructions.

Type Field Definition

If Data Segment (S = 1, E = 0)

If Code Segment (S = 1, E = 1)

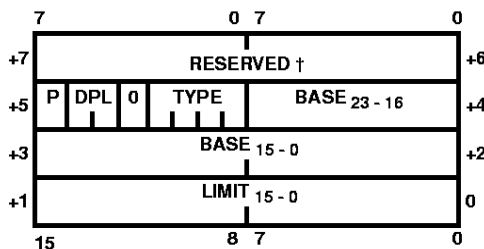
Data segments ( $S = 1, E = 0$ ) may be either read-only or read-write as controlled by the  $W$  bit of the access rights byte. Read-only ( $W = 0$ ) data segments may not be written into. Data segments may grow in two directions, as determined by the Expansion Direction (ED) bit: upwards ( $ED = 0$ ) 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 ( $S = 0, \text{Type} = 1-3$ )

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 ( $S = 0, \text{Type} = 4-7$ )

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.

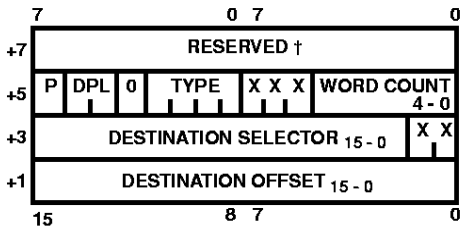
Exception 13 is generated when the gate is used if a destination selector does not refer to the correct descriptor type. The word count field is used in the call gate descriptor to indicate the number of parameters (0-31 words) to be automatically copied from the caller's stack to the stack of the called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

**Segment Descriptor Cache Registers**

A segment descriptor cache register is assigned to each of the four segment registers (CS, SS, DS, ES). Segment descriptors are automatically loaded (cached) into a segment descriptor cache register (Figure 12) whenever the associated segment register is loaded with a selector.

Only segment descriptors may be loaded into segment descriptor cache registers. Once loaded, all references to that segment of memory use the cached descriptor information instead of reaccessing the descriptor. The descriptor cache registers are not visible to programs. No instructions exist to store their contents. They only change when a segment register is loaded.



† MUST BE SET TO 0 FOR COMPATIBILITY WITH FUTURE UPGRADES

FIGURE 11. GATE DESCRIPTOR

TABLE 12. GATE DESCRIPTOR FORMAT FIELD

NAME	VALUE	DESCRIPTION
TYPE	4	Call Gate
	5	Task Gate
	6	Interrupt Gate
	7	Trap Gate
P	0	Descriptor Contents are not valid
	1	Descriptor Contents are valid
DPL	0 - 3	Descriptor Privilege Level
WORD COUNT	0 - 31	Number of words to copy from callers stack to called procedures stack. Only used with call gate.
DESTINATION SELECTOR	16-Bit Selector	Selector to the target code segment (call, interrupt or selector Trap Gate). Selector to the target task state segment (Task Gate).
DESTINATION OFFSET	16-Bit Offset	Entry point within the target code segment

**Selector Fields**

A protected mode selector has three fields: descriptor entry index, local or global descriptor table indicator (TI), and selector privilege (RPL) as shown in Figure 13. These fields select one of two memory based tables of descriptors, select the appropriate table entry and allow high-speed testing of the selector's privilege attribute (refer to privilege discussion below).

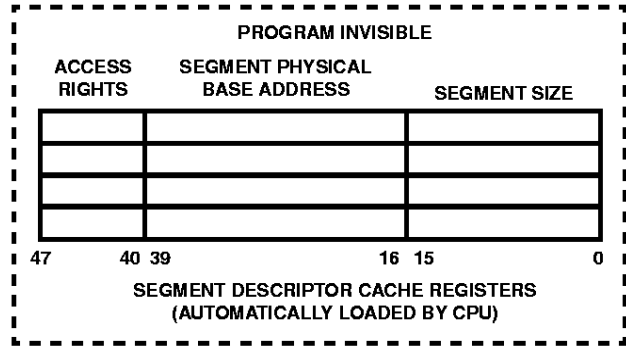
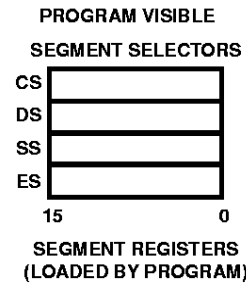
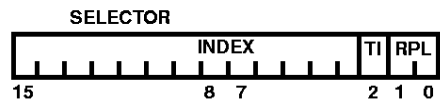


FIGURE 12. DESCRIPTOR CACHE REGISTERS



BITS	NAME	FUNCTION
1 - 0	Requested Privilege Level (RPL)	Indicates Selector Privilege Level Desired
2	Table Indicator (TI)	TI = 0 Use Global Descriptor Table (GDT) TI = 1 Use Local Descriptor Table (LDT)
15 - 3	Index	Select Descriptor Entry In Table

FIGURE 13. SELECTOR FIELDS

**Local and Global Descriptor Tables**

Two tables of descriptors, called descriptor tables, contain all descriptors accessible by a task at any given time. A descriptor table is a linear array of up to 8192 descriptors. The upper 13 bits of the selector value are an index into a descriptor table. Each table has a 24-bit base register to locate the descriptor table in physical memory and a 16-bit limit register that confine descriptor access to the defined limits of the table as shown in Figure 14. A restartable exception (13) will occur if an attempt is made to reference a descriptor outside the table limits.

One table, called the Global Descriptor table (GDT), contains descriptors available to all tasks. The other table, called the Local Descriptor Table (LDT), contains descriptors that can be private to a task. Each task may have its own private LDT. The GDT may contain all descriptor types except interrupt and trap descriptors. The LDT may contain only segment, task gate, and call gate descriptors. A segment cannot be accessed by a task if its segment descriptor does not exist in either descriptor table at the time of access.

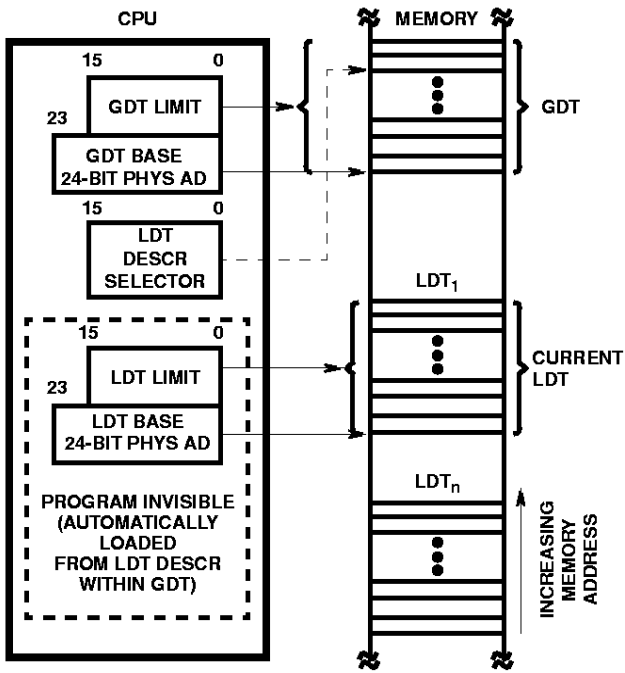
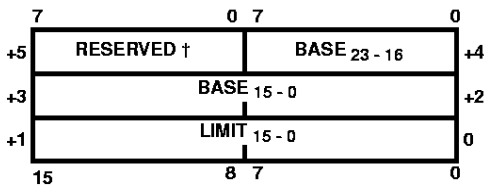


FIGURE 14. LOCAL AND GLOBAL DESCRIPTOR TABLE DEFINITION

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.

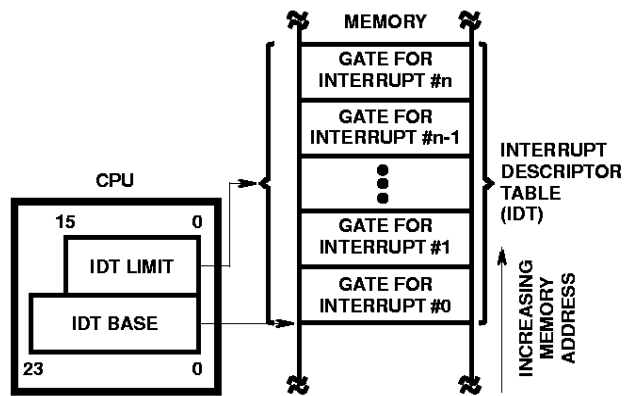
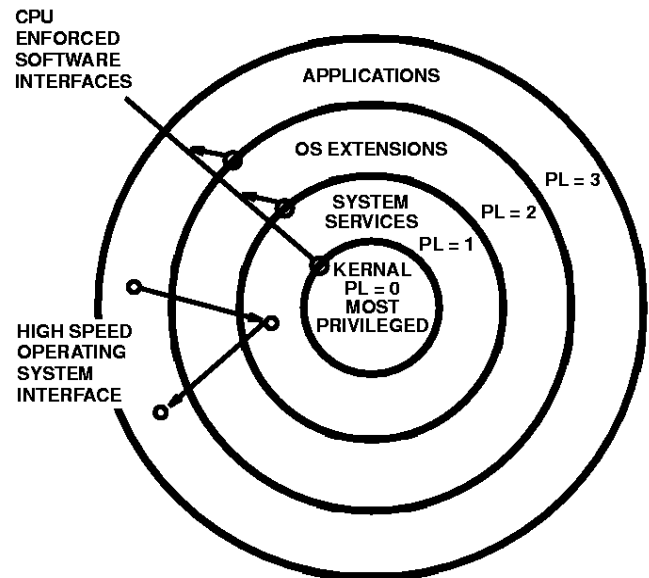


FIGURE 16. INTERRUPT DESCRIPTOR TABLE DEFINITION

**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

## Task Privilege

A task always executes at one of the four privilege levels. The task privilege level at any specific instant is called the Current Privilege Level (CPL) and is defined by the lower two bits of the CS register. CPL cannot change during execution in a single code segment. A task's CPL may only be changed by control transfers through gate descriptors to a new code segment (See Control Transfer). Tasks begin executing at the CPL value specified by the code segment selector within TSS when the task is initiated via a task switch operation (See Figure 18). A task executing at Level 0 can access all data segments defined in the GDT and the task's LDT and is considered the most trusted level. A task executing a Level 3 has the most restricted access to data and is considered the least trusted level.

## Descriptor Privilege

Descriptor privilege is specified by the Descriptor Privilege Level (DPL) field of the descriptor access byte. DPL specifies the least trusted task privilege level (CPL) at which a task may access the descriptor. Descriptors with DPL = 0 are the most protected. Only tasks executing at privilege level 0 (CPL = 0) may access them. Descriptors with DPL = 3 are the least protected (i.e. have the least restricted access) since tasks can access them when CPL = 0, 1, 2, or 3). This rule applies to all descriptors, except LDT descriptors.

## Selector Privilege

Selector privilege is specified by the Requested Privilege Level (RPL) field in the least significant two bits of a selector. Selector RPL may establish a less trusted privilege level than the current privilege level for the use of a selector. This level is called the task's effective privilege level (EPL). RPL can only reduce the scope of a task's access to data with this selector. A task's effective privilege is the numeric maximum of RPL and CPL. A selector with RPL = 0 imposes no additional restriction on its use while a selector with RPL = 3 can only refer to segments at privilege Level 3 regardless of the task's CPL. RPL is generally used to verify that pointer

parameters passed to a more trusted procedure are not allowed to use data at a more privileged level than the caller (refer to pointer testing instructions).

## Descriptor Access and Privilege Validation

Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL. The two basic types of segment accesses are control transfer (selectors loaded into CS) and data (selectors loaded into DS, ES or SS).

## Data Segment Access

Instructions that load selectors into DS and ES must refer to a data segment descriptor or readable code segment descriptor. The CPL of the task and the RPL of the selector must be the same as or more privileged (numerically equal to or lower than) than the descriptor DPL. In general, a task can only access data segments at the same or less privileged levels than the CPL or RPL (whichever is numerically higher) to prevent a program from accessing data it cannot be trusted to use.

An exception to the rule is a readable conforming code segment. This type of code segment can be read from any privilege level.

If the privilege checks fail (e.g. DPL is numerically less than the maximum of CPL and RPL) or an incorrect type of descriptor is referenced (e.g. gate descriptor or execute only code segment) exception 13 occurs. If the segment is not present, exception 11 is generated.

Instructions that load selectors into SS must refer to data segment descriptors for writable data segments. The descriptor privilege (DPL) and RPL must equal CPL. All other descriptor types or a privilege level violation will cause exception 13. A not present fault causes exception 12.

TABLE 13. DESCRIPTOR TYPES USED FOR CONTROL TRANSFER

CONTROL TRANSFER TYPES	OPERATION TYPES	DESCRIPTOR REFERENCED	DESCRIPTOR TABLE
Intersegment within the same privilege levels	JMP, CALL, RET, IRET (Note 4)	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET (Note 4)	Code Segment	GDT/LDT
Task Switch	CALL, JMP	Task State Segment	GDT
	CALL, JMP	Task Gate	GDT/LDT
	IRET (Note 5) Interrupt Instruction, Exception External Interrupt	Task Gate	IDT

### NOTES:

4. NT (Nested Task bit of flag word) = 0
5. NT (Nested Task bit of flag word) = 1

### Control Transfer

Four types of control transfer can occur when a selector is loaded into CS by a control transfer operation (see Table 13). Each transfer type can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules (e.g. JMP through a call gate or RET to a Task State Segment) will cause exception 13.

The ability to reference a descriptor for control transfer is also subject to rules of privilege. A CALL or JUMP instruction may only reference a code segment descriptor with DPL equal to the task CPL or a conforming segment with DPL of equal or greater privilege than CPL. The RPL of the selector used to reference the code descriptor must have as much privilege as CPL.

RET and IRET instructions may only reference code segment descriptors with descriptor privilege equal to or less privileged than the task CPL. The selector loaded into CS is the return address from the stack. After the return, the selector RPL is the task's new CPL. If CPL changes, the old stack pointer is popped after the return address.

When a JMP or CALL references a Task State Segment descriptor, the descriptor DPL must be the same or less privileged than the task's CPL. Reference to a valid Task State Segment descriptor causes a task switch (see Task Switch Operation). Reference to a Task State Segment descriptor at a more privileged level than the task's CPL generates exception 13.

When an instruction or interrupt references a gate descriptor, the gate DPL must have the same or less privilege than the task CPL. If DPL is at a more privileged level than CPL, exception 13 occurs. If the destination selector contained in the gate references a code segment descriptor, the code segment descriptor DPL must be the same or more privileged than the task CPL. If not, Exception 13 is issued. After the control transfer, the code segment descriptors DPL is the task's new CPL. If the destination selector in the gate references a task state segment, a task switch is automatically performed (see Task Switch Operation).

The privilege rules on control transfer require:

- JMP or CALL direct to a code segment (code segment descriptor) can only be a conforming segment with DPL of equal or greater privilege than CPL or a non-conforming segment at the same privilege level.
- Interrupts within the task, or calls that may change privilege levels, can only transfer control through a gate at the same or a less privileged level than CPL to a code segment at the same or more privileged level than CPL.
- Return instructions that don't switch tasks can only return control to a code segment at the same or less privileged level.
- Task switch can be performed by a call, jump or interrupt which references either a task gate or task state segment at the same or less privileged level.

### Privilege Level Changes

Any control transfer that changes CPL within the task, causes a change of stacks as part of the operation. Initial values of SS:SP for privilege levels 0, 1, and 2 are kept in the task state segment (refer to Task Switch Operation). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and SP registers and the previous stack pointer is pushed onto the new stack.

When returning to the original privilege level, its stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words, as specified in the gate, are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

### Protection

The 80C286 includes mechanisms to protect critical instructions that effect the CPU execution state (e.g. HLT) and code or data segments from improper usage. These protection mechanisms are grouped into three forms:

- Restricted usage of segments (e.g. no write allowed to read-only data segments). The only segments available for use are defined by descriptors in the Local Descriptor Table (LDT) and Global Descriptor Table (GDT).
- Restricted access to segments via the rules of privilege and descriptor usage.
- Privileged instructions or operations that may only be executed at certain privilege levels as determined by the CPL and I/O Privilege Level (IOPL). The IOPL is defined by bits 14 and 13 of the flag word.

These checks are performed for all instructions and can be split into three categories: segment load checks (Table 14), operand reference checks (Table 15), and privileged instruction checks (Table 16). Any violation of the rules shown will result in an exception. A not-present exception related to the stack segment causes exception 12.

**TABLE 14. SEGMENT REGISTER LOAD CHECKS**

ERROR DESCRIPTION	EXCEPTION NUMBER
Descriptor table limit exceeded	13
Segment descriptor not-present	11 or 12
Privilege rules violated	13
Invalid descriptor/segment type segment register load: - Read only data segment load to SS - Special control descriptor load to DS, ES, SS - Execute only Segment load to DS, ES, SS - Data segment load to CS - Read/Execute code segment load SS	13

TABLE 15. OPERAND REFERENCE CHECKS

ERROR DESCRIPTION	EXCEPTION NUMBER
Write into code segment	13
Read from execute-only code segment	13
Write to read-only data segment	13
Segment limit exceeded (See Note)	12 or 13

NOTE: Carry out in offset calculations is ignored.

TABLE 16. PRIVILEGED INSTRUCTION CHECKS

ERROR DESCRIPTION	EXCEPTION NUMBER
CPL $\neq$ 0 when executing the following instructions: LIDT, LLDT, LGDT, LTR, LMSW, CTS, HLT	13
CPT > IOPL when executing the following instructions: INS, IN, OUTS, OUT, STI, CLI, LOCK	13

The IRET and POPF instructions do not perform some of their defined functions if CPL is not of sufficient privilege (numerically small enough). Precisely these are:

- The IF bit is not changed if CPL is greater than IOPL.
- The IOPL field of the flag word is not changed if CPL is greater than 0.

No exceptions or other indication are given when these conditions occur.

### Exceptions

The 80C286 detects several types of exceptions and interrupts in protected mode (see Table 17). Most are restartable after the exceptional condition is removed. Interrupt handlers for most exceptions can read an error code, pushed on the stack after the return address, that identifies the selector involved (0 if none). The return address normally points to the failing instruction including all leading prefixes. For a processor extension segment overrun exception, the return

address will not point at the ESC instruction that caused the exception; however, the processor extension registers may contain the address of the failing instruction.

These exceptions indicate a violation to privilege rules or usage rules has occurred. Restart is generally not attempted under those conditions.

All these checks are performed for all instructions and can be split into three categories: segment load checks (Table 14), operand reference checks (Table 15), and privileged instruction checks (Table 16). Any violation of the rules shown will result in an exception. A not-present exception causes exception 11 or 12 and is restartable.

### SPECIAL OPERATIONS

#### Task Switch Operation

The 80C286 provides a built-in task switch operation which saves the entire 80C286 execution state (registers, address space, and a link to the previous task), loads a new execution state, and commences execution in the new task. Like gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS) or task gate descriptor in the GDT or LDT. An INT instruction, exception, or external interrupt may also invoke the task switch operation by selecting a task gate descriptor in the associated IDT descriptor entry.

The TSS descriptor points at a segment (see Figure 18) containing the entire 80C286 execution state while a task gate descriptor contains a TSS selector. The limit field of the descriptor must be greater than 002B(H).

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 80C286 called the Task Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector. The IRET instruction is used to return control to the task that called the current task or was interrupted. Bit 14 in the flag register is called the Nested Task (NT) bit. It controls the

TABLE 17. PROTECTED MODE EXCEPTIONS

INTERRUPT VECTOR	FUNCTION	RETURN ADDRESS AT FALLING INSTRUCTION?	ALWAYS RESTARTABLE?	ERROR CODE ON STACK?
8	Double exception detected	Yes	No (Note 7)	Yes
9	Processor extension segment overrun	No	No (Note 7)	No
10	Invalid task state segment	Yes	Yes	Yes
11	Segment not present	Yes	Yes	Yes
12	Stack segment overrun or stack segment not present	Yes	Yes (Note 6)	Yes
13	General protection	Yes	No (Note 7)	Yes

NOTES:

6. When a PUSH or POP instruction attempts to wrap around the stack segment, the machine state after the exception will not be restartable because stack segment wrap around is not permitted. This condition is identified by the value of the saved SP being either 0000(H), 0001(H), FFFE(H), or FFFF(H).
7. These exceptions indicate a violation to privilege rules or usage rules has occurred. Restart is generally not attempted under those conditions.

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.



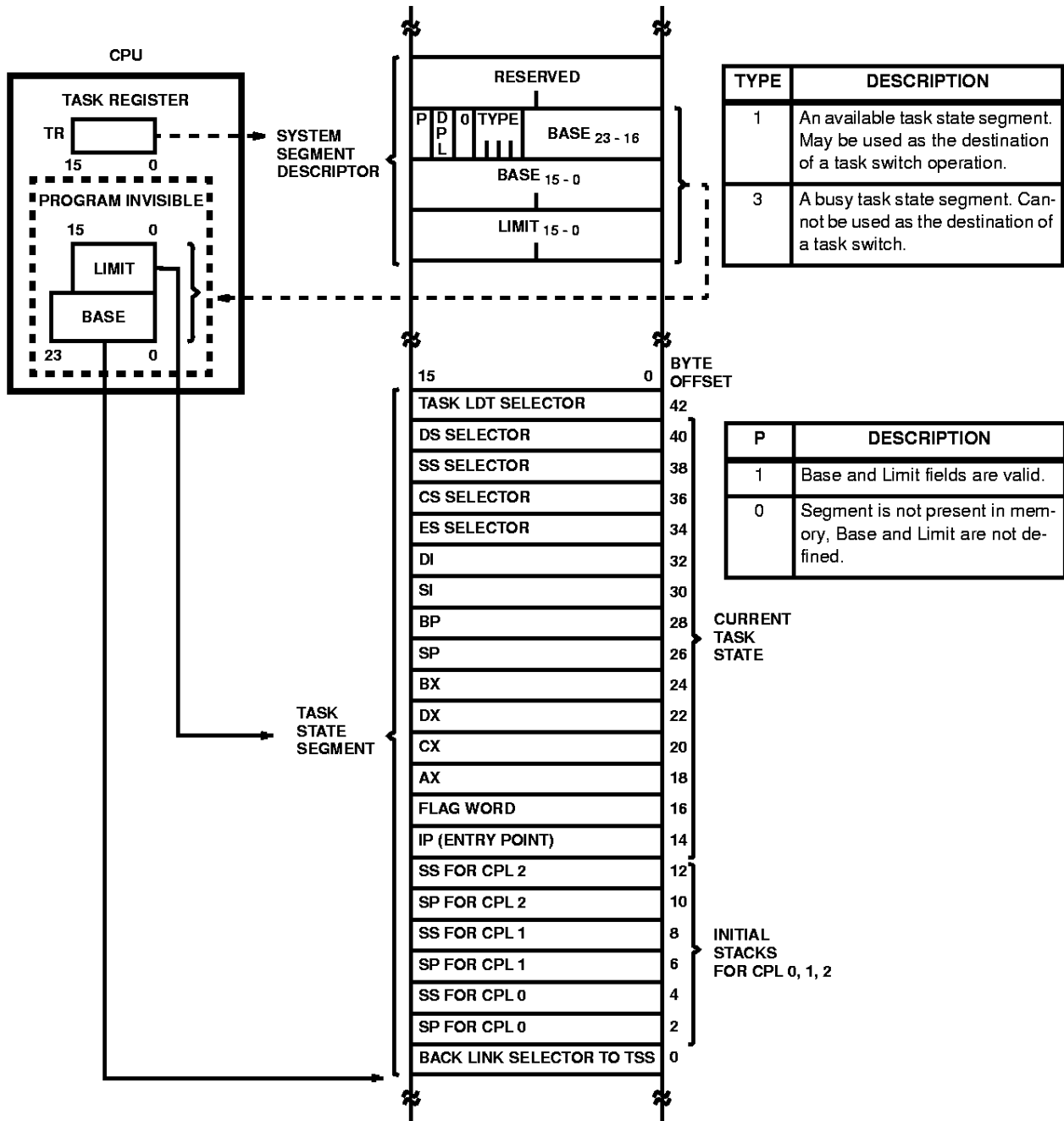


FIGURE 18. TASK STATE SEGMENT AND TSS REGISTERS

**System Interface**

The 80C286 system interface appears in two forms: a local bus and a system bus. The local bus consists of address, data, status, and control signals at the pins of the CPU. A system bus is any buffered version of the local bus. A system bus may also differ from the local bus in terms of coding of status and control lines and/or timing and loading of signals.

**Bus Interface Signals and Timing**

The 80C286 microsystems local bus interfaces the 80C286 to local memory and I/O components. The interface has 24 address lines, 16 data lines, and 8 status and control signals.

The 80C286 CPU, 82C284 clock generator, 82C288 bus controller, 82289 bus arbiter, 82C86H/87H transceivers, and 82C82/83H latches provide a buffered and decoded system bus interface. The 82C284 generates the system clock and

synchronizes  $\overline{\text{READY}}$  and RESET. The 82C288 converts bus operation status encoded by the 80C286 into command and bus control signals. The 82289 bus arbiter generates Multibus™ bus arbitration signals. These components can provide the critical timing required for most system bus interfaces including the Multibus.

### Bus Hold Circuitry

To avoid high current conditions caused by floating inputs to CMOS devices, and to eliminate the need for pull-up/down resistors, "bus-hold" circuitry has been used on the 80C286 pins 4-6, 36-51 and 66-68 (See Figure 19A and 19B). The circuit shown in Figure 19A will maintain the last valid logic state if no driving source is present (i.e. an unconnected pin or a driving source which goes to a high impedance state). The circuit shown in Figure 19B will maintain a high impedance logic one state if no driving source is present. To override the "bus-hold" circuits, an external driver must be capable of sinking or sourcing approximately 400 microamps at valid input voltage levels. Since this "bus-hold" circuitry is active and not a "resistive" type element, the associated power supply current is negligible, and power dissipation is significantly reduced when compared to the use of passive pull-up resistors.

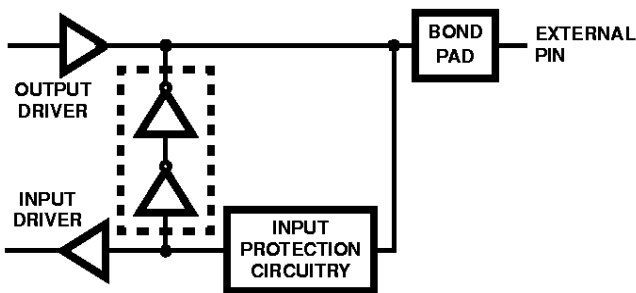


FIGURE 19A. BUS HOLD CIRCUITRY, PINS 36-51, 66, 67

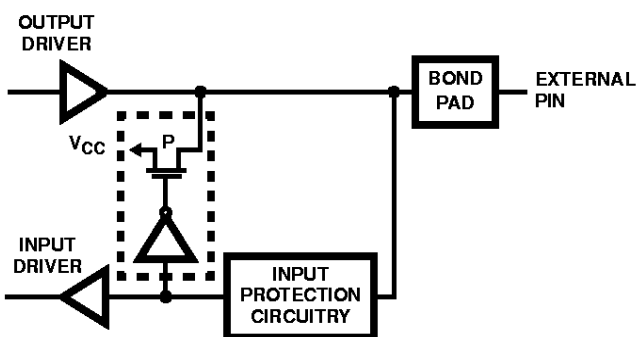


FIGURE 19B. BUS HOLD CIRCUITRY, PINS 4-6, 68

### Physical Memory and I/O Interface

A maximum of 16 megabytes of physical memory can be addressed in protected mode. One megabyte can be addressed in real address mode. Memory is accessible as bytes or words. Words consist of any two consecutive bytes addressed with the least significant byte stored in the lowest

address. Byte transfers occur on either half of the 16-bit local data bus. Even bytes are accessed over  $D_{7-0}$  while odd bytes are transferred over  $D_{15-8}$ . Even addressed words are transferred over  $D_{15-0}$  in one bus cycle, while odd addressed word require two bus operations. The first transfers data on  $D_{15-8}$ , and the second transfers data on  $D_{7-0}$ . Both byte data transfers occur automatically, transparent to software.

Two bus signals,  $A_0$  and  $\overline{\text{BHE}}$ , control transfers over the lower and upper halves of the data bus. Even address byte transfers are indicated by  $A_0$  LOW and  $\overline{\text{BHE}}$  HIGH. Odd address byte transfers are indicated by  $A_0$  HIGH and  $\overline{\text{BHE}}$  LOW. Both  $A_0$  and  $\overline{\text{BHE}}$  are LOW for even address word transfers.

The I/O address space contains 64K addresses in both modes. The I/O space is accessible as either bytes or words, as is memory. Byte wide peripheral devices may be attached to either the upper or lower byte of the data bus. Byte-wide I/O devices attached to the upper data byte ( $D_{15-8}$ ) are accessed with odd I/O addresses. Devices on the lower data byte are accessed with even I/O addresses. An interrupt controller such as Harris's 82C59A must be connected to the lower data byte ( $D_{7-0}$ ) for proper return of the interrupt vector.

### Bus Operation

The 80C286 uses a double frequency system clock (CLK input) to control bus timing. All signals on the local bus are measured relative to the system CLK input. The CPU divides the system clock by 2 to produce the internal processor clock, which determines bus state. Each processor clock is composed of two system clock cycles named phase 1 and phase 2. The 82C284 clock generator output (PCLK) identifies the next phase of the processor clock. (See Figure 20.)

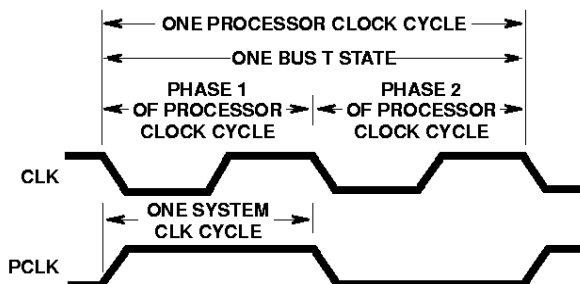


FIGURE 20. SYSTEM AND PROCESSOR CLOCK RELATIONSHIPS

Six types of bus operations are supported; memory read, memory write, I/O read, I/O write, interrupt acknowledge, and halt/shutdown. Data can be transferred at a maximum rate of one word per two processor clock cycles.

The 80C286 bus has three basic states: idle ( $T_I$ ), send status ( $T_S$ ), and perform command ( $T_C$ ). The 80C286 CPU also has a fourth local bus state called hold ( $T_H$ ).  $T_H$  indicates that the 80C286 has surrendered control of the local bus to another bus master in response to a HOLD request.

Each bus state is one processor clock long. Figure 21 shows the four 80C286 local bus states and allowed transitions.

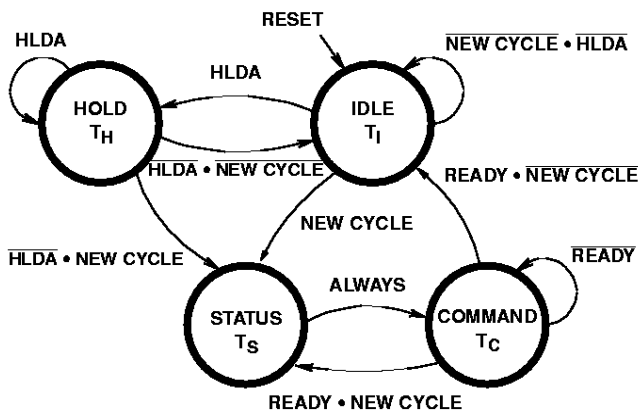


FIGURE 21. 80C286 BUS STATES

### Bus States

The idle ( $T_I$ ) state indicates that no data transfers are in progress or requested. The first active state  $T_S$  is signaled by status line  $S_1$  or  $S_0$  going LOW and identifying phase 1 of the processor clock. During  $T_S$ , the command encoding, the address, and data (for a write operation) are available on the 80C286 output pins. The 82C288 bus controller decodes the status signals and generates Multibus compatible read/write command and local transceiver control signals.

After  $T_S$ , the perform command ( $T_C$ ) state is entered. Memory or I/O devices respond to the bus operation during  $T_C$ , either transferring read data to the CPU or accepting write data.  $T_C$  states may be repeated as often as necessary to ensure sufficient time for the memory or I/O device to respond. The  $READY$  signal determines whether  $T_C$  is repeated. A repeated  $T_C$  state is called a wait state.

During hold ( $T_H$ ), the 80C286 will float all address, data, and status output drivers enabling another bus master to use the local bus. The 80C286 HOLD input signal is used to place the 80C286 into the  $T_H$  state. The 80C286 HLDA output signal indicates that the CPU has entered  $T_H$ .

### Pipelined Addressing

The 80C286 uses a local bus interface with pipelined timing to allow as much time as possible for data access. Pipelined timing allows a new bus operation to be initiated every two processor cycles, while allowing each individual bus operation to last for three processor cycles.

The timing of the address outputs is pipelined such that the address of the next bus operation becomes available during the current bus operation. Or, in other words, the first clock of the next bus operation is overlapped with the last clock of the current bus operation. Therefore, address decode and routing logic can operate in advance of the next bus operation.

External address latches may hold the address stable for the entire bus operation, and provide additional AC and DC buffering.

The 80C286 does not maintain the address of the current bus operation during all  $T_C$  states. Instead, the address for the next bus operation may be emitted during phase 2 of any  $T_C$ .

The address remains valid during phase 1 of the first  $T_C$  to guarantee hold time, relative to ALE, for the address latch inputs.

### Bus Control Signals

The 82C288 bus controller provides control signals; address latch enable (ALE), Read/Write commands, data transmit/receive (DT/ $\bar{R}$ ), and data enable (DEN) that control the address latches, data transceivers, write enable, and output enable for memory and I/O systems.

The Address Latch Enable (ALE) output determines when the address may be latched. ALE provides at least one system CLK period of address hold time from the end of the previous bus operation until the address for the next bus operation appears at the latch outputs. This address hold time is required to support Multibus and common memory systems.

The data bus transceivers are controlled by 82C288 outputs Data Enable (DEN) and Data Transmit/Receive (DT/ $\bar{R}$ ). DEN enables the data transceivers; while DT/ $\bar{R}$  controls transceiver direction. DEN and DT/ $\bar{R}$  are timed to prevent bus contention between the bus master, data bus transceivers, and system data bus transceivers.

### Command Timing Controls

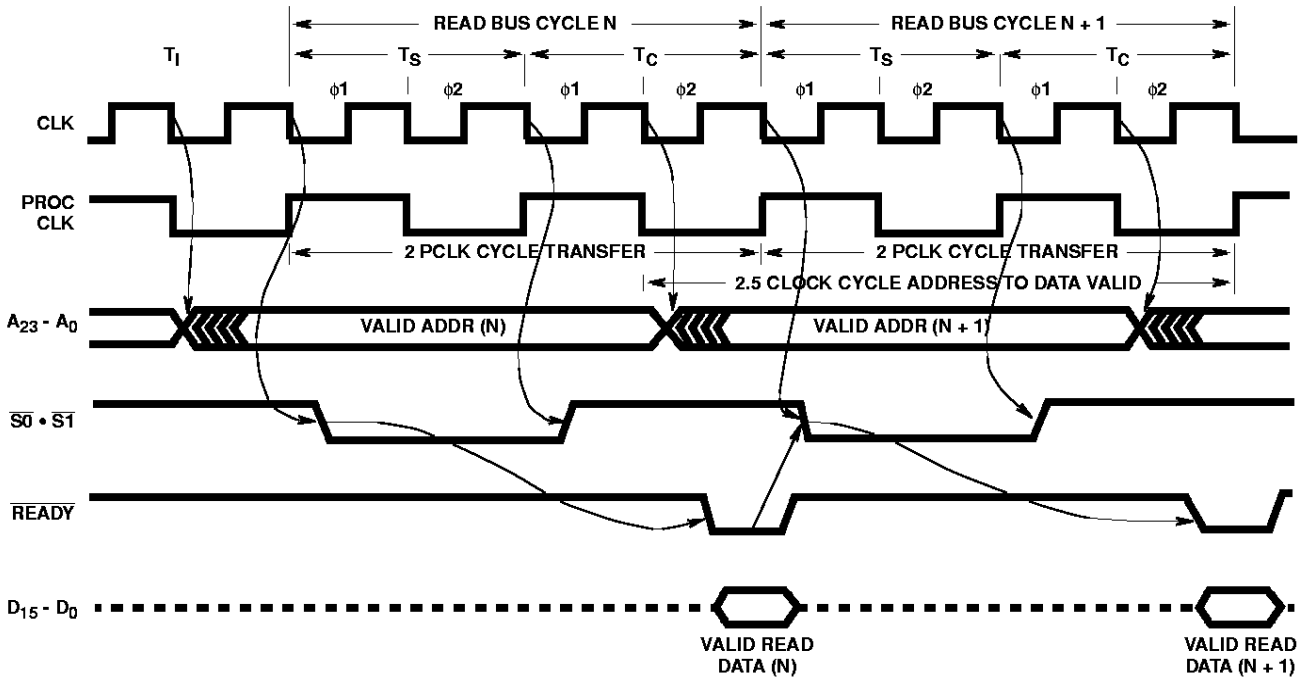
Two system timing customization options, command extension and command delay, are provided on the 80C286 local bus.

Command extension allows additional time for external devices to respond to a command and is analogous to inserting wait states on the 80C86. External logic can control the duration of any bus operation such that the operation is only as long as necessary. The  $READY$  input signal can extend any bus operation for as long as necessary.

Command delay allows an increase of address or write data setup time to system bus command active for any bus operation by delaying when the system bus command becomes active. Command delay is controlled by the 82C288 CMDLY input. After  $T_S$ , the bus controller samples CMDLY at each falling edge of CLK. If CMDLY is HIGH, the 82C288 will not activate the command signal. When CMDLY is LOW, the 82C288 will activate the command signal. After the command becomes active, the CMDLY input is not sampled.

When a command is delayed, the available response time from command active to return read data or accept write data is less. To customize system bus timing, an address decoder can determine which bus operations require delaying the command. The CMDLY input does not affect the timing of ALE, DEN or DT/ $\bar{R}$ .

Figure 23 illustrates four uses of CMDLY. Example 1 shows delaying the read command two system CLKs for cycle N-1 and no delay for cycle N, and example 2 shows delaying the read command one system CLK for cycle N-1 and one system CLK delay for cycle N.



PIPELINING: VALID ADDRESS (N + 1) AVAILABLE IN LAST PHASE OF BUS CYCLE (N).

FIGURE 22. BASIC BUS CYCLE

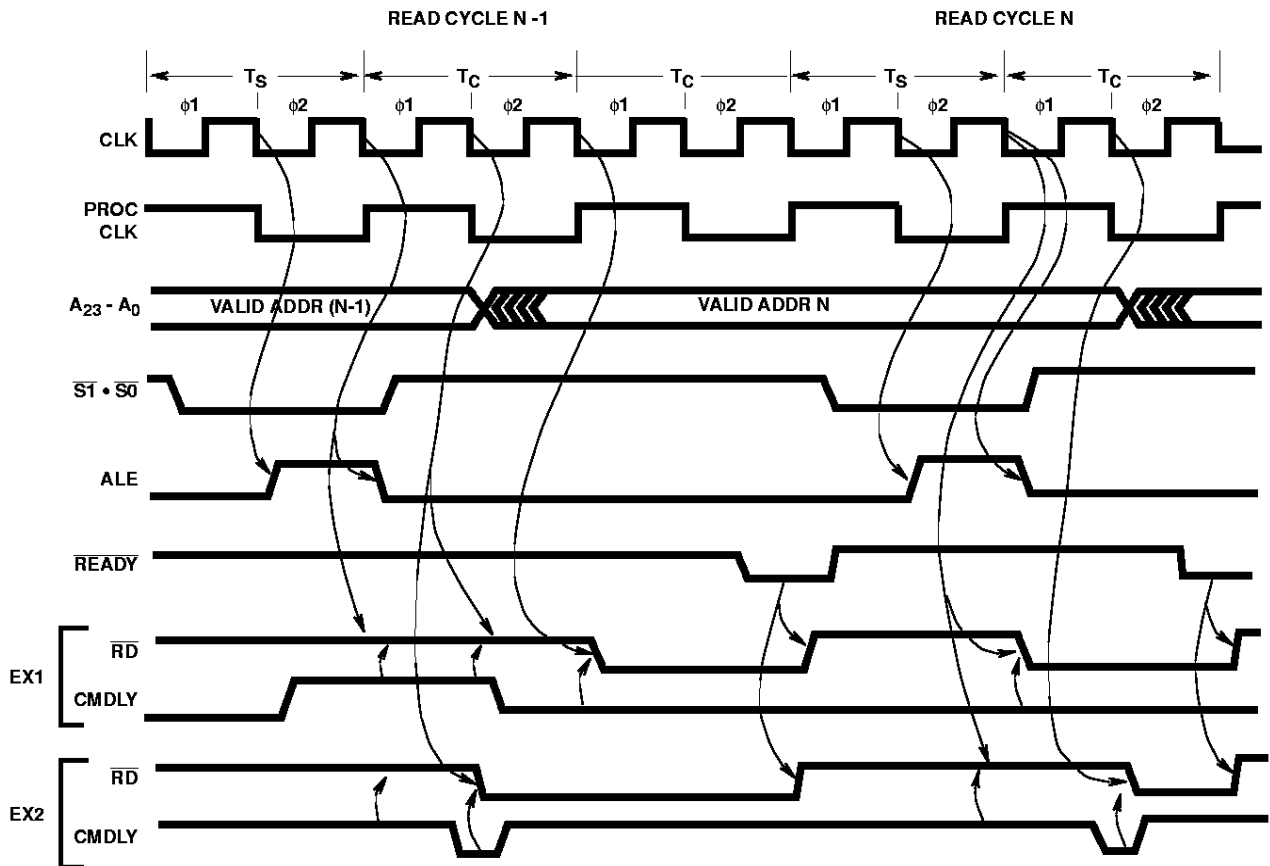


FIGURE 23. CMDLY CONTROLS THE LEADING EDGE OF COMMAND SIGNAL

### 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 (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 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 (ARDY) is designed to accept such signals. The 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  $\overline{SRDY}$ .

### Data Bus Control

Figures 25, 26, and 27 show how the  $DT/\overline{R}$ , DEN, data bus, and address signals operate for different combinations of read, write, and idle bus operations.  $DT/\overline{R}$  goes active (LOW) for a read operation.  $DT/\overline{R}$  remains HIGH before, during, and between write operations.

The data bus is driven with write data during the second phase of  $T_S$ . The delay in write data timing allows the read data drivers, from a previous read cycle, sufficient time to enter three-state OFF before the 80C286 CPU begins driving the local data bus for write operations. Write data will always remain valid for one system clock past the last  $T_C$  to provide sufficient hold time for Multibus or other similar memory or I/O systems. During write-read or write-idle sequences the data bus enters a high impedance state during the second phase of the processor cycle after the last  $T_C$ . In a write-write sequence the data bus does not enter a high impedance state between  $T_C$  and  $T_S$ .

### Bus Usage

The 80C286 local bus may be used for several functions: instruction data transfers, data transfers by other bus masters, instruction fetching, processor extension data transfers, interrupt acknowledge, and halt/shutdown. This section describes local bus activities which have special signals or requirements. Note that I/O transfers take place in exactly the same manner as memory transfers (i.e. to the 80C286 the timing, etc. of an I/O transfer is identical to a memory transfer).

### HOLD and HLDA

HOLD and HLDA allow another bus master to gain control of the local bus by placing the 80C286 bus into the  $T_H$  state. The sequence of events required to pass control between the 80C286 and another local bus master are shown in Figure 28.

In this example, the 80C286 is initially in the  $T_H$  state as signaled by HLDA being active. Upon leaving  $T_H$ , as signaled by HLDA going inactive, a write operation is started. During the write operation another local bus master requests the local bus from the 80C286 as shown by the HOLD signal. After completing the write operation, the 80C286 performs one  $T_1$  bus cycle, to guarantee write data hold time, then enters  $T_H$  as signaled by HLDA going active.

The  $\overline{CMDLY}$  signal and  $\overline{ARDY}$  ready are used to start and stop the write bus command, respectively. Note that  $\overline{SRDY}$  must be inactive or disabled by  $\overline{SRDYEN}$  to guarantee  $\overline{ARDY}$  will terminate the cycle.

HOLD must not be active during the time from the leading edge of RESET until 34 CLKs following the trailing edge of RESET unless the 80C286 is in the Halt condition. To ensure that the 80C286 remains in the Halt condition until the processor Reset operation is complete, no interrupts should occur after the execution of HLT until 34 CLKs after the trailing edge of the RESET pulse.

### LOCK

The CPU asserts an active lock signal during Interrupt-Acknowledge cycles, the XCHG instruction, and during some descriptor accesses. Lock is also asserted when the LOCK prefix is used. The LOCK prefix may be used with the following ASM-286 assembly instructions; MOVS, INS and OUTS. For bus cycles other than Interrupt-Acknowledge cycles, Lock will be active for the first and subsequent

cycles of a series of cycles to be locked. Lock will not be shown active during the last cycle to be locked. For the next-to-last cycle, Lock will become inactive at the end of the first  $T_C$  regardless of the number of wait states inserted. For Interrupt-Acknowledge cycles, Lock will be active for each cycle, and will become inactive at the end of the first  $T_C$  for each cycle regardless of the number of wait-states inserted.

**Instruction Fetching**

The 80C286 Bus Unit (BU) will fetch instructions ahead of the current instruction being executed. This activity is called prefetching. It occurs when the local bus would otherwise be idle and obeys the following rules:

A prefetch bus operation starts when at least two bytes of the 6-byte prefetch queue are empty.

The prefetcher normally performs word prefetches independent of the byte alignment of the code segment base in physical memory.

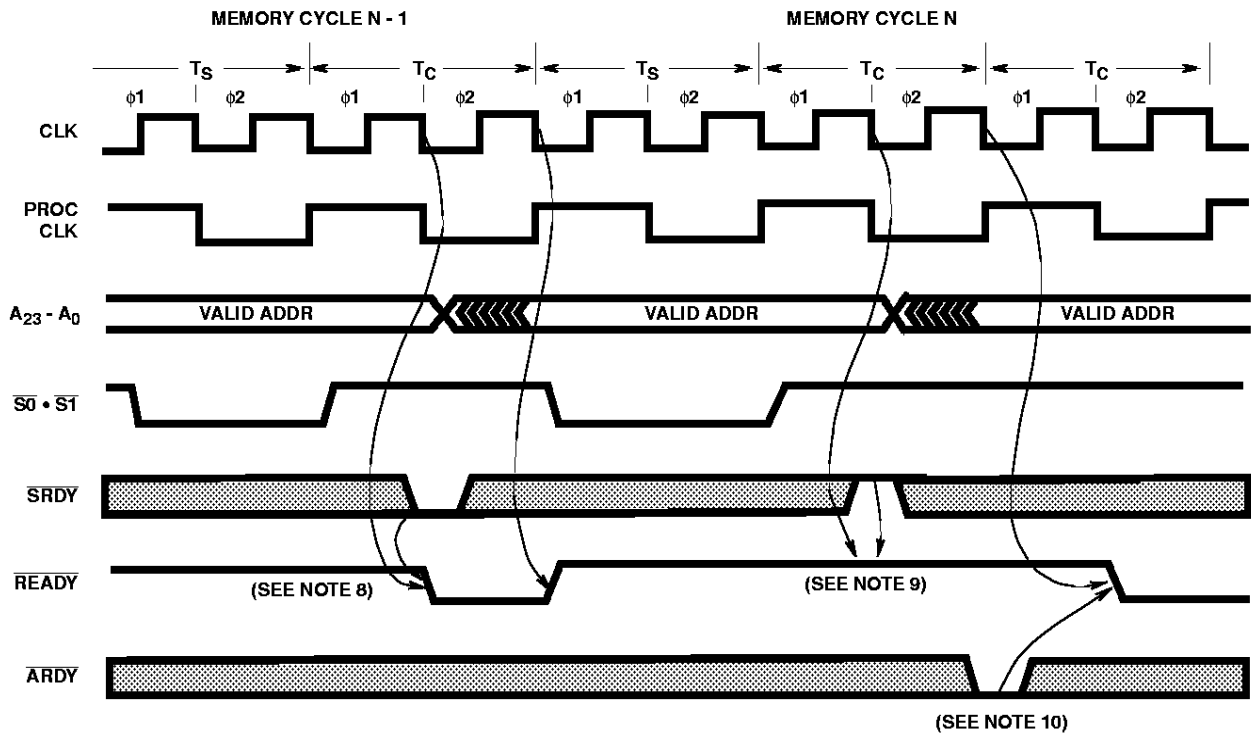
The prefetcher will perform only a byte code fetch operation for control transfers to an instruction beginning on a numerically odd physical address.

Prefetching stops whenever a control transfer or HLT instruction is decoded by the IU and placed into the instruction queue.

In real address mode, the prefetcher may fetch up to 6 bytes beyond the last control transfer or HLT instruction in a code segment.

In protected mode, the prefetcher will never cause a segment overrun exception. The prefetcher stops at the last physical memory word of the code segment. Exception 13 will occur if the program attempts to execute beyond the last full instruction in the code segment.

If the last byte of a code segment appears on an even physical memory address, the prefetcher will read the next physical byte of memory (perform a word code fetch). The value of this byte is ignored and any attempt to execute it causes exception 13.



- NOTES:
- 8. SRDYEN is active low.
  - 9. If SRDYEN is high, the state of SRDY will not effect READY.
  - 10. ARDYEN is active low.

**FIGURE 24. SYNCHRONOUS AND ASYNCHRONOUS READY**

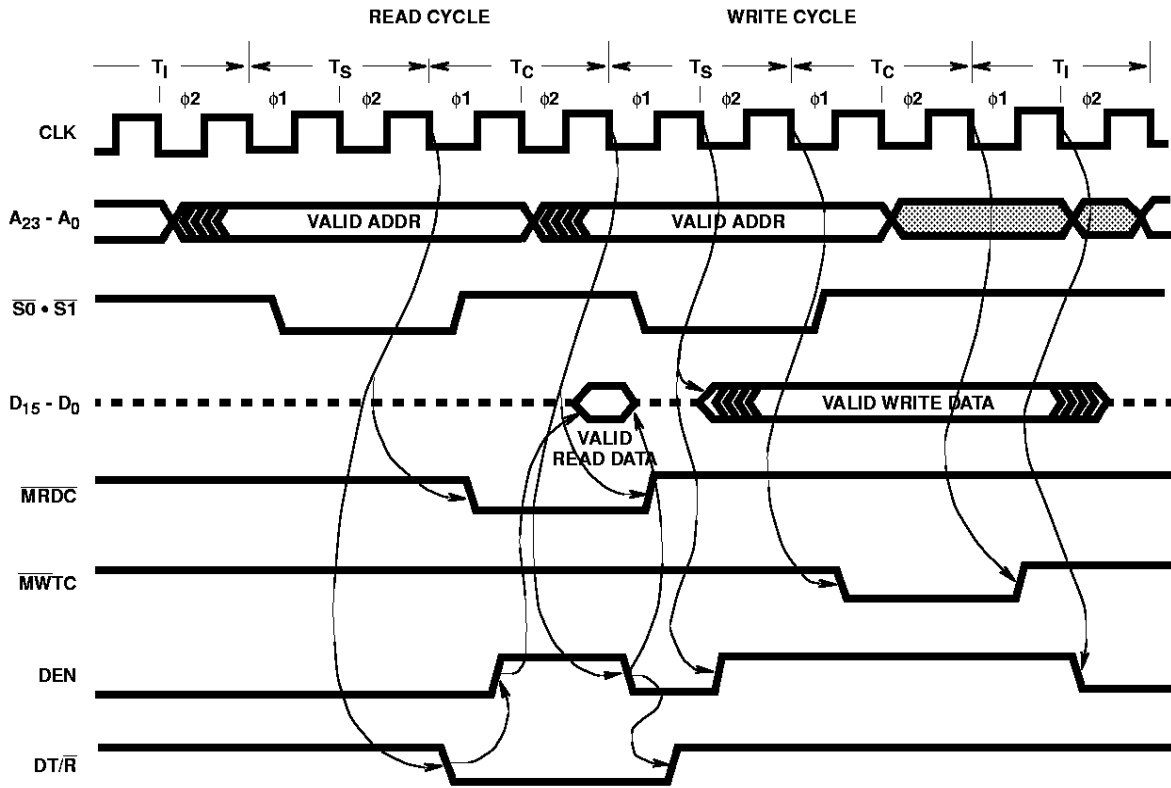


FIGURE 25. BACK TO BACK READ-WRITE CYCLE

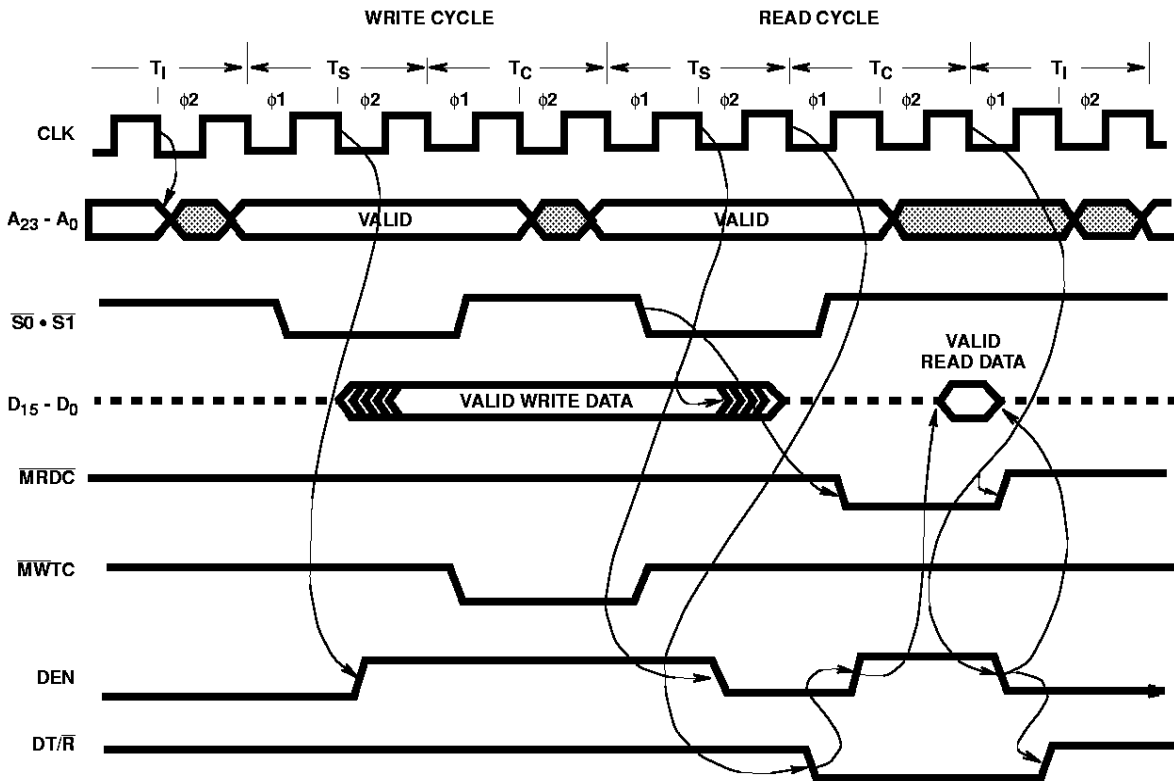


FIGURE 26. BACK TO BACK WRITE-READ CYCLE

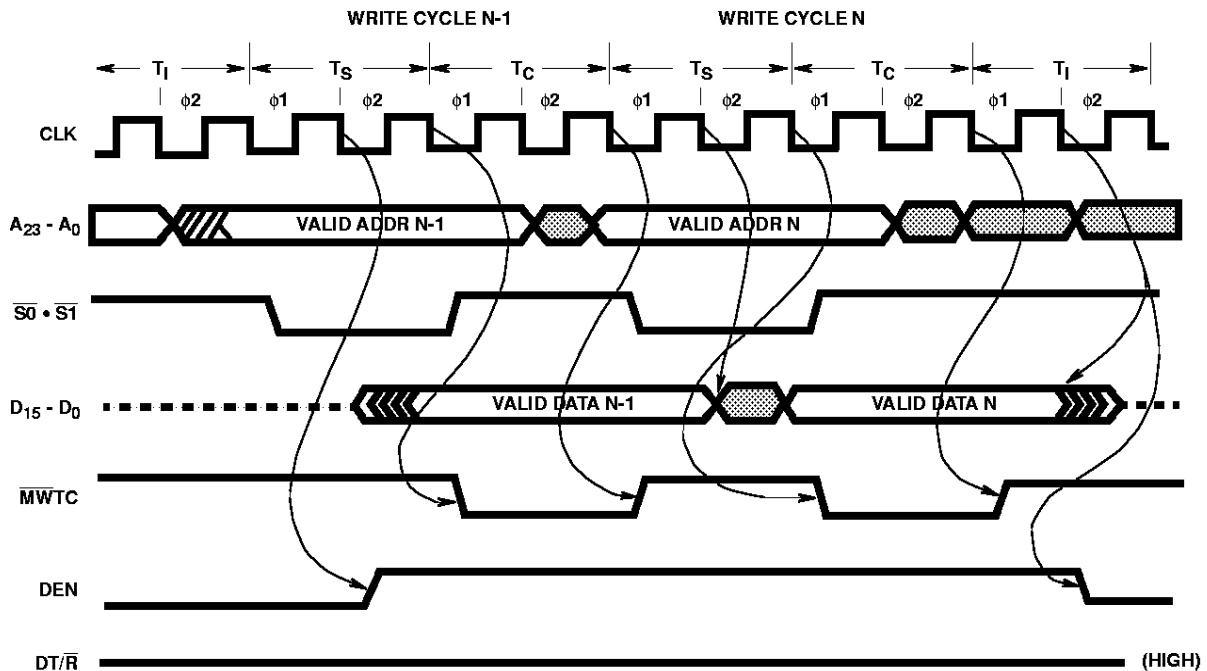


FIGURE 27. BACK TO BACK WRITE-WRITE CYCLE

### Processor Extension Transfers

The processor extension interface uses I/O port addresses 00F8(H), and 00FC(H) which are part of the I/O port address range reserved by Harris. An ESC instruction with Machine Status Word bits EM = 0 and  $T_S = 0$  will perform I/O bus operations to one or more of these I/O port addresses independent of the value of IOPL and CPL.

ESC instructions with memory references enable the CPU to accept PEREQ inputs for processor extension operand transfers. The CPU will determine the operand starting address and read/write status of the instruction. For each operand transfer, two or three bus operations are performed, one word transfer with I/O port address 00FA(H) and one or two bus operations with memory. Three bus operations are required for each word operand aligned on an odd byte address.

### Interrupt Acknowledge Sequence

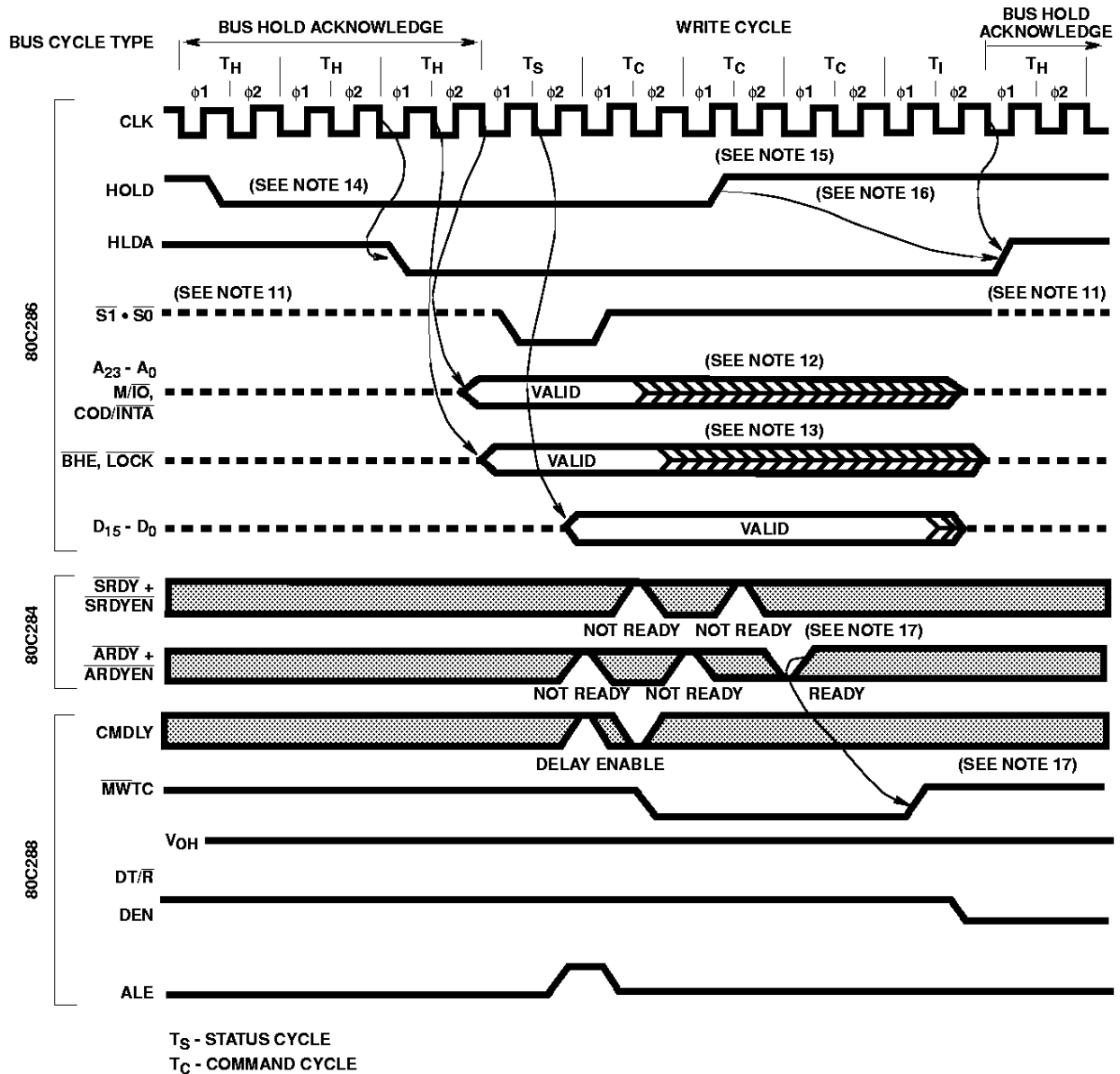
Figure 29 illustrates an interrupt acknowledge sequence performed by the 80C286 in response to an INTR input. An interrupt acknowledge sequence consists of two INTA bus operations. The first allows a master 82C59A Programmable Interrupt Controller (PIC) to determine which if any of its slaves should return the interrupt vector. An eight bit vector is read on D<sub>0</sub>-D<sub>7</sub> of the 80C286 during the second INTA bus operation to select an interrupt handler routine from the interrupt table.

The Master Cascade Enable (MCE) signal of the 82C288 is used to enable the cascade address drivers during INTA bus operations (See Figure 29) onto the local address bus for distribution to slave interrupt controllers via the system address bus. The 80C286 emits the  $\overline{LOCK}$  signal (active LOW) during  $T_S$  of the first INTA bus operation. A local bus "hold" request will not be honored until the end of the second INTA bus operation.

Three idle processor clocks are provided by the 80C286 between INTA bus operations to allow for the minimum INTA to INTA time and CAS (cascade address) out delay of the 82C59A. The second INTA bus operation must always have at least one extra  $T_C$  state added via logic controlling  $\overline{READY}$ . A<sub>23</sub>-A<sub>0</sub> are in three-state OFF until after the first  $T_C$  state of the second INTA bus operation. This prevents bus contention between the cascade address drivers and CPU address drivers. The extra  $T_C$  state allows time for the 80C286 to resume driving the address lines for subsequent bus operations.



# 80C286



$T_S$  - STATUS CYCLE  
 $T_C$  - COMMAND CYCLE

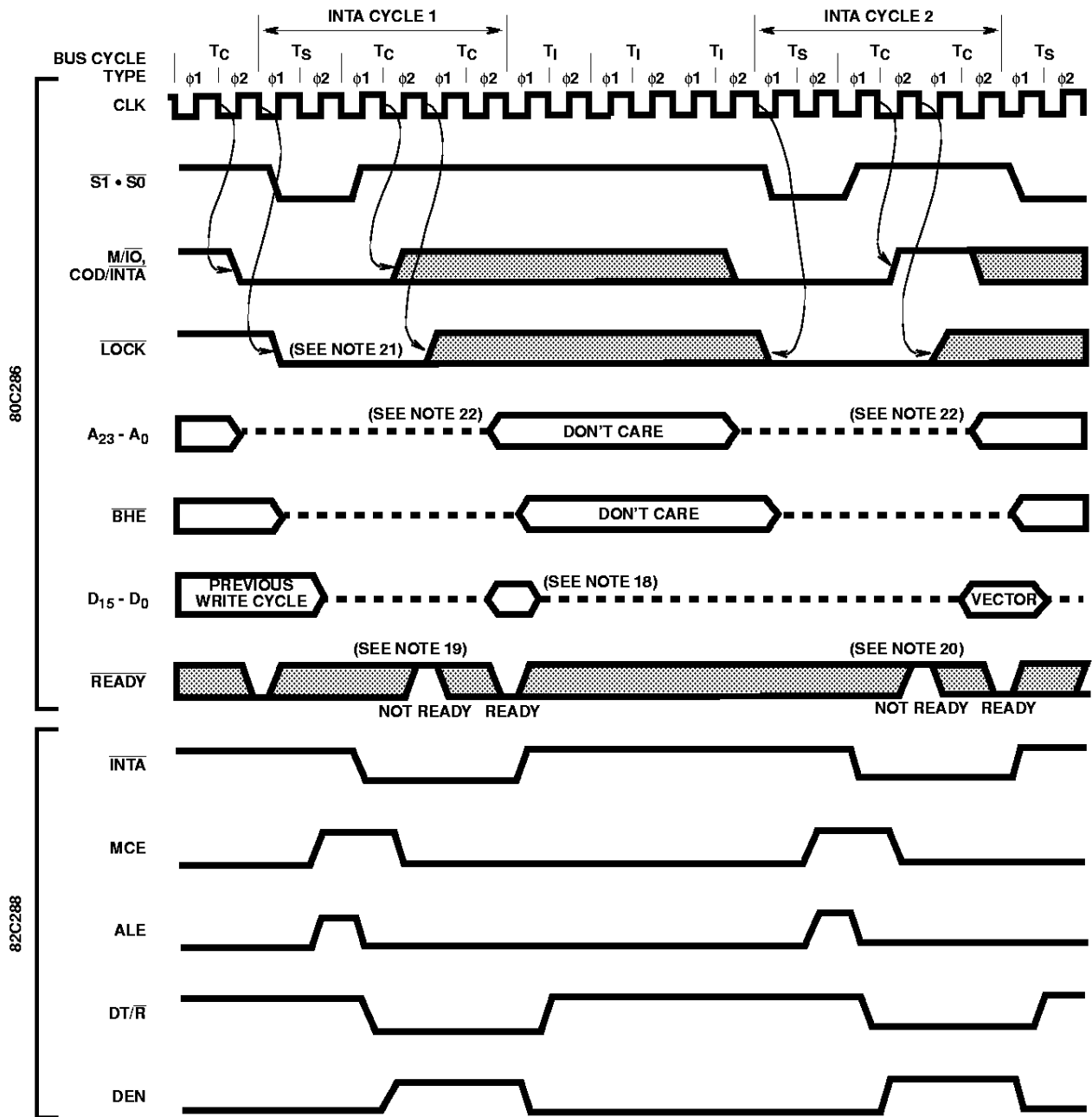
## NOTES:

- Status lines are held at a high impedance logic one by the 80C286 during a HOLD state.
- Address,  $\overline{M/I\overline{O}}$  and  $\overline{COD/INTA}$  may start floating during any  $T_C$  depending on when internal 80C286 bus arbiter decides to release bus to external HOLD. The float starts in  $\phi 2$  of  $T_C$ .
- $\overline{BHE}$  and  $\overline{LOCK}$  may start floating after the end of any  $T_C$  depending on when internal 80C286 bus arbiter decides to release bus to external HOLD. The float starts in  $\phi 1$  of  $T_C$ .
- The minimum HOLD to HLDA time is shown. Maximum is one  $T_H$  longer.
- The earliest HOLD time is shown. It will always allow a subsequent memory cycle if pending is shown.
- The minimum HOLD to HLDA time is shown. Maximum is a function of the instruction, type of bus cycle and other machine state (i.e., Interrupts, Waits, Lock, etc.).
- Asynchronous ready allows termination of the cycle. Synchronous ready does not signal ready in this example. Synchronous ready state is ignored after ready is signaled via the asynchronous input.

**FIGURE 28. MULTIBUS WRITE TERMINATED BY ASYNCHRONOUS READY WITH BUS HOLD**



# 80C286



## 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<sub>23</sub>-A<sub>0</sub>, BHE, and LOCK until after the first T<sub>C</sub> state. The CPU imposed one/clock delay prevents has contention between cascade address buffer being disabled by MCE ↓ 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. LOCK is active for the first INTA cycle to prevent the 82289 from releasing the bus between INTA cycles in a multi-master system. LOCK is also active for the second INTA cycle.
23. A<sub>23</sub>-A<sub>0</sub> exits three-state OFF during  $\phi 2$  of the second T<sub>C</sub> in the INTA cycle.

FIGURE 29. INTERRUPT ACKNOWLEDGE SEQUENCE

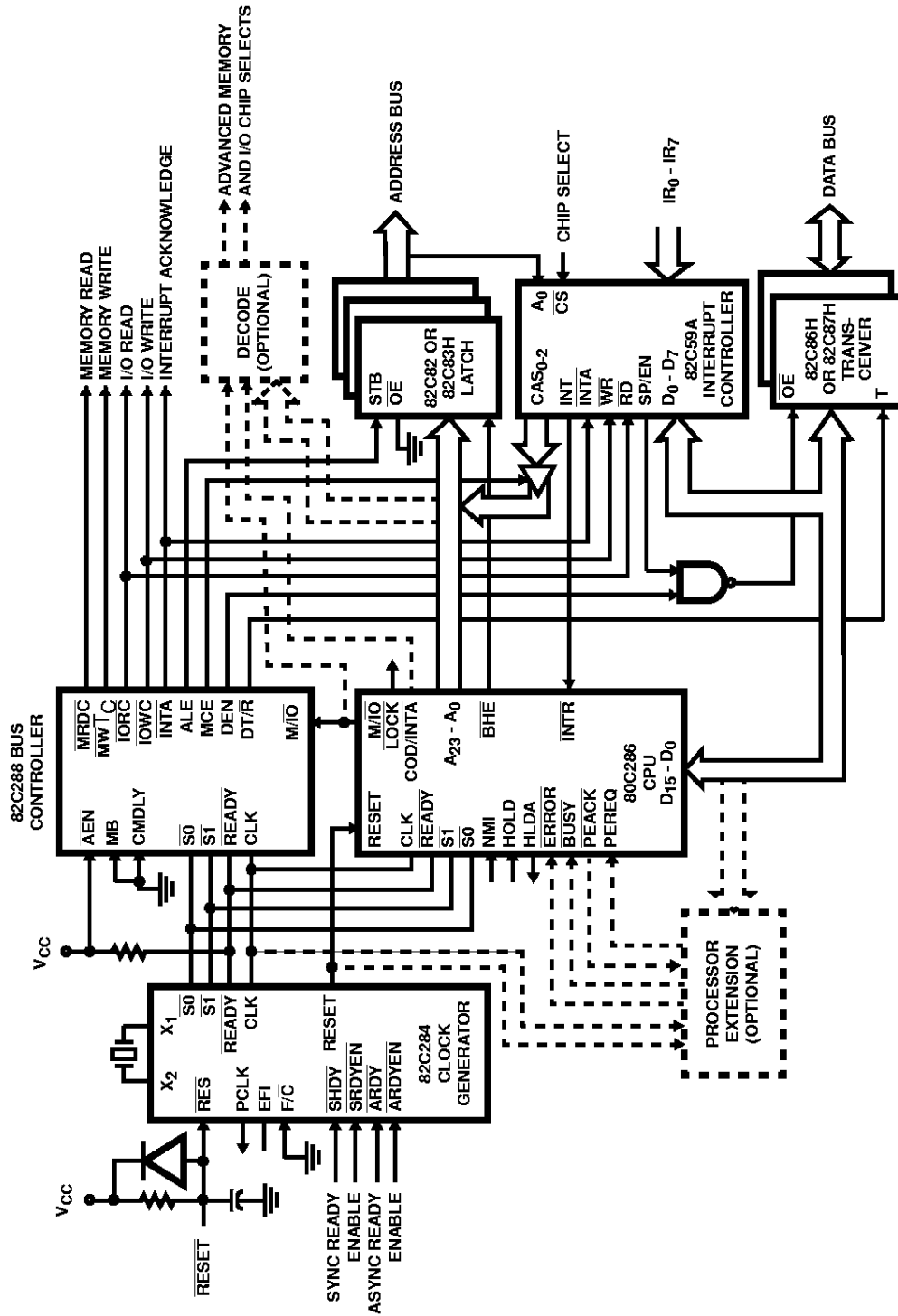


FIGURE 30. BASIC 80C286 SYSTEM CONFIGURATION

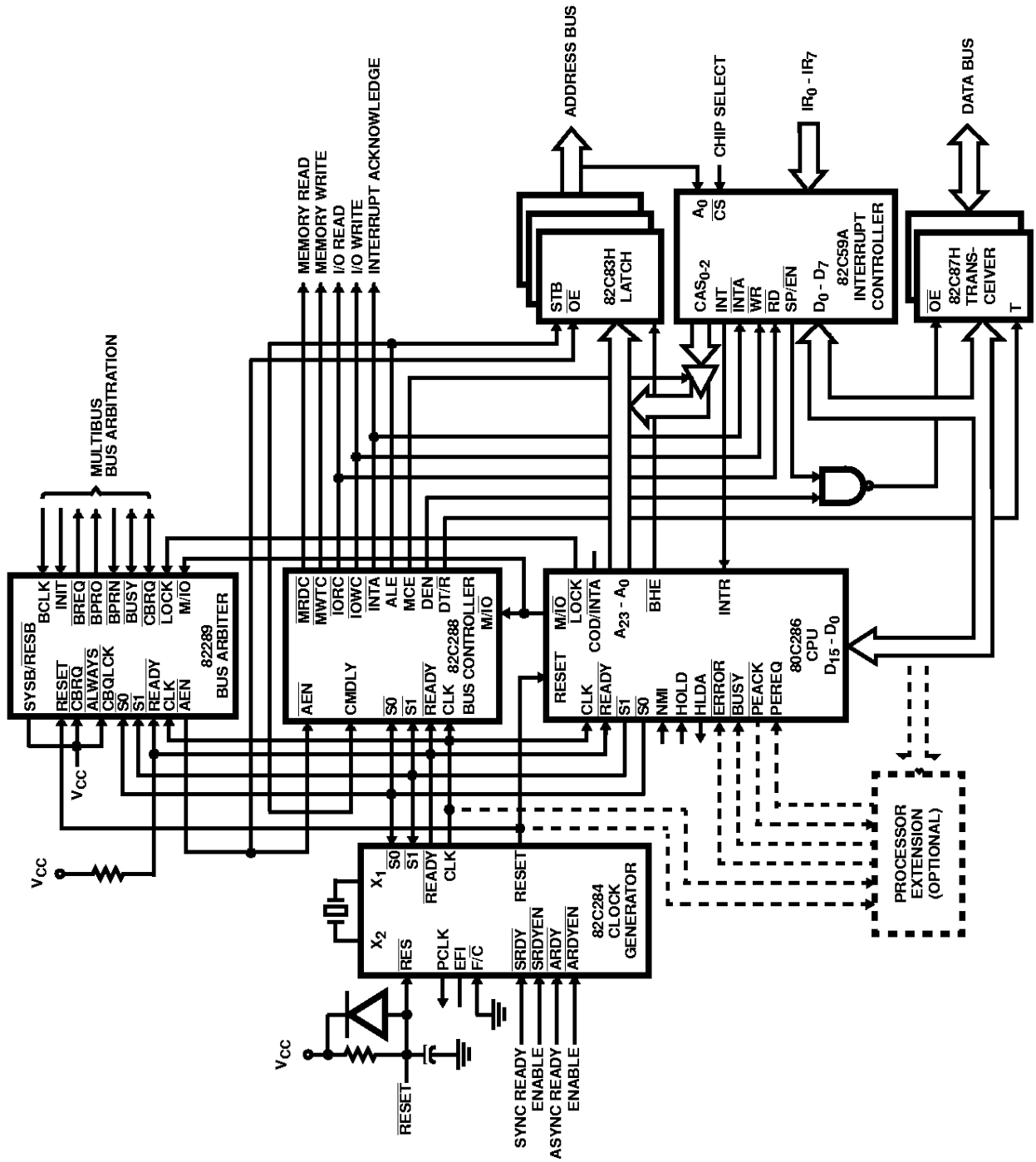


FIGURE 31. MULTIBUS SYSTEM BUS INTERFACE

# 80C286

## Absolute Maximum Ratings

Supply Voltage .....	±8.0V
Input, Output or I/O Voltage Applied .....	GND -1.0V to $V_{CC} + 1.0V$
Storage Temperature Range .....	-65°C to +150°C
Junction Temperature, PGA .....	+175°C
PLCC .....	+150°C
Lead Temperature (Soldering, 10s) .....	+300°C
(PLCC - Lead Tips Only)	

## Thermal Information

Thermal Resistance (Typical)	$\theta_{JA}$ (°C/W)	$\theta_{JC}$ (°C/W)
PDIP Package .....	35	6
CERDIP Package .....	33	9
Maximum Package Power Dissipation		
PGA Package .....	1.22W	
PLCC Package .....	2.2W	
Gate Count .....	22,500	

*CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.*

## Operating Conditions

Operating Voltage Range	Operating Temperature Range
80C286-10, -12 .....	180C286-10, -12, -16, -20 .....
80C286-16, -20, -25 .....	C80C286-12, -16, -20, -25 .....
+4.5V to +5.5V	-40°C to +85°C
+4.75V to +5.25V	0°C to +70°C

**DC Electrical Specifications**  $V_{CC} = +5V \pm 10\%$ ,  $T_A = 0^\circ C$  to  $+70^\circ C$  (C80C286-12),  $V_{CC} = +5V \pm 5\%$ ,  $T_A = 0^\circ C$  to  $+70^\circ C$  (C80C286-16, -20, -25),  $V_{CC} = +5V \pm 10\%$ ,  $T_A = -40^\circ C$  to  $+85^\circ C$  (180C286-10, -12),  $V_{CC} = +5V \pm 5\%$ ,  $T_A = -40^\circ C$  to  $+85^\circ C$  (180C286-16, -20)

SYMBOL	PARAMETER	MIN	MAX	UNITS	TEST CONDITIONS
$V_{IL}$	Input LOW Voltage	-0.5	0.8	V	
$V_{IH}$	Input HIGH Voltage	2.0	$V_{CC} + 0.5$	V	
$V_{ILC}$	CLK Input LOW Voltage	-0.5	1.0	V	
$V_{IHC}$	CLK Input HIGH Voltage	3.6	$V_{CC} + 0.5$	V	
$V_{OL}$	Output LOW Voltage	-	0.4	V	$I_{OL} = 2.0mA$
$V_{OH}$	Output HIGH Voltage	3.0 $V_{CC} - 0.4$	- -	V	$I_{OH} = -2.0mA$ , $I_{OH} = -100\mu A$
$I_I$	Input Leakage Current	-10	10	$\mu A$	$V_{IN} = GND$ or $V_{CC}$ Pins 29, 31, 57, 59, 61, 63-64
$I_{SH}$	Input Sustaining Current on $\overline{BUSY}$ and ERROR Pins	-30	-500	$\mu A$	$V_{IN} = GND$ (See Note 28)
$I_{BHL}$	Input Sustaining Current LOW	38	200	$\mu A$	$V_{IN} = 1.0V$ (See Note 24)
$I_{BHH}$	Input Sustaining Current HIGH	-50	-400	$\mu A$	$V_{IN} = 3.0V$ (See Note 25)
$I_O$	Output Leakage Current	-10	10	$\mu A$	$V_O = GND$ or $V_{CC}$ Pins 1, 7-8, 10-28, 32-34
$I_{CCOP}$	Active Power Supply Current	-	185	mA	80C286-10 (See Note 27)
		-	220	mA	80C286-12 (See Note 27)
		-	260	mA	80C286-16 (See Note 27)
		-	310	mA	80C286-20 (See Note 27)
		-	410	mA	80C286-25 (See Note 27)
$I_{CCSB}$	Standby Power Supply Current	-	5	mA	(See Note 26)

**Capacitance**  $T_A = +25^\circ C$ , All Measurements Referenced to Device GND

SYMBOL	PARAMETER	TYP	UNITS	TEST CONDITIONS
$C_{CLK}$	CLK Input Capacitance	10	pF	FREQ = 1MHz
$C_{IN}$	Other Input Capacitance	10	pF	
$C_{I/O}$	I/O Capacitance	10	pF	

### NOTES:

24.  $I_{BHL}$  should be measured after lowering  $V_{IN}$  to GND and then raising to 1.0V on the following pins: 36-51, 66, 67.
25.  $I_{BHH}$  should be measured after raising  $V_{IN}$  to  $V_{CC}$  and then lowering to 3.0V on the following pins: 4-6, 36-51, 66-68.
26.  $I_{CCSB}$  tested with the clock stopped in phase two of the processor clock cycle.  $V_{IN} = V_{CC}$  or GND,  $V_{CC} = V_{CC} (Max)$ , outputs unloaded.
27.  $I_{CCOP}$  measured at 10MHz for the 80C286-10, 12.5MHz for the 80C286-12, 16MHz for the 80C286-16, 20MHz for the 80C286-20, and 25MHz for the 80C286-25.  $V_{IN} = 2.4V$  or 0.4V,  $V_{CC} = V_{CC} (Max)$ , outputs unloaded.
28.  $I_{SH}$  should be measured after raising  $V_{IN}$  to  $V_{CC}$  and then lowering to GND on pins 53 and 54.

## 80C286

**AC Electrical Specifications**  $V_{CC} = +5V \pm 10\%$ ,  $T_A = 0^\circ C$  to  $+70^\circ C$  (C80C286-12),  $T_A = -40^\circ C$  to  $+85^\circ C$  (I80C286-10, -12)  
 $V_{CC} = +5V \pm 5\%$ ,  $T_A = 0^\circ C$  to  $+70^\circ C$  (C80C286-16),  $T_A = -40^\circ C$  to  $+85^\circ C$  (I80C286-16) AC Timings  
 are Referenced to 0.8V and 2.0V Points of the Signals as Illustrated in Data Sheet Waveforms,  
 Unless Otherwise Specified

SYMBOL	PARAMETER	10MHz		12.5MHz		16MHz		UNIT	TEST CONDITION
		MIN	MAX	MIN	MAX	MIN	MAX		
<b>TIMING REQUIREMENTS</b>									
1	System Clock (CLK) Period	50	-	40	-	31	-	ns	
2	System Clock (CLK) LOW Time	12	-	11	-	7	-	ns	At 1.0V
3	System Clock (CLK) HIGH Time	16	-	13	-	11	-	ns	At 3.6V
17	System Clock (CLK) RISE Time	-	8	-	8	-	5	ns	1.0V to 3.6V
18	System Clock (CLK) FALL Time	-	8	-	8	-	5	ns	3.6V to 1.0V
4	Asynchronous Inputs SETUP Time	20	-	15	-	5	-	ns	(Note 29)
5	Asynchronous Inputs HOLD Time	20	-	15	-	5	-	ns	(Note 29)
6	RESET SETUP Time	19	-	10	-	10	-	ns	
7	RESET HOLD Time	0	-	0	-	0	-	ns	
8	Read Data SETUP Time	8	-	5	-	5	-	ns	
9	Read Data HOLD Time	4	-	4	-	3	-	ns	
10	$\overline{READY}$ SETUP Time	26	-	20	-	12	-	ns	
11	$\overline{READY}$ HOLD Time	25	-	20	-	5	-	ns	
20	Input RISE/FALL Times	-	10	-	8	-	6	ns	0.8V to 2.0V
<b>TIMING RESPONSES</b>									
12A	Status/ $\overline{PEACK}$ Active Delay	1	22	1	21	1	18	ns	1, (Notes 31, 35)
12B	Status/ $\overline{PEACK}$ Inactive Delay	1	30	1	24	1	20	ns	1, (Notes 31, 34)
13	Address Valid Delay	1	35	1	32	1	27	ns	1, (Notes 30, 31)
14	Write Data Valid Delay	0	40	0	31	0	28	ns	1, (Notes 30, 31)
15	Address/Status/Data Float Delay	0	47	0	32	0	29	ns	2, (Note 33)
16	HLDA Valid Delay	0	47	0	25	0	25	ns	1, (Notes 31, 36)
19	Address Valid to Status SETUP Time	27	-	22	-	16	-	ns	1, (Notes 31, 32)

**NOTES:**

- 29. Asynchronous inputs are INTR, NMI, HOLD, PEREQ, ERROR, and BUSY. This specification is given only for testing purposes, to assure recognition at a specific CLK edge.
- 30. Delay from 1.0V on the CLK to 0.8V or 2.0V.
- 31. Output load:  $C_L = 100pF$ .
- 32. Delay measured from address either reaching 0.8V or 2.0V (valid) to status going active reaching 0.8V or status going inactive reaching 2.0V.
- 33. Delay from 1.0V on the CLK to Float (no current drive) condition.
- 34. Delay from 1.0V on the CLK to 0.8V for min. (HOLD time) and to 2.0V for max. (inactive delay).
- 35. Delay from 1.0V on the CLK to 2.0V for min. (HOLD time) and to 0.8V for max. (active delay).
- 36. Delay from 1.0V on the CLK to 2.0V.

**AC Test Conditions**

TEST CONDITION	$I_L$ (CONSTANT CURRENT SOURCE)	$C_L$
1	[2.0mA]	100pF
2	-6mA ( $V_{OH}$ to Float) 8mA ( $V_{OL}$ to Float)	100pF

## 80C286

**AC Electrical Specifications**  $V_{CC} = +5V \pm 5\%$ ,  $T_A = 0^\circ C$  to  $+70^\circ C$  (C80C286-20, -25),  $T_A = -40^\circ C$  to  $+85^\circ C$  (I80C286-20)  
 AC Timings are Referenced to the 1.5V Point of the Signals as Illustrated in Data Sheet Waveforms,  
 Unless Otherwise Specified

SYMBOL	PARAMETER	20MHz		25MHz		UNIT	TEST CONDITION
		MIN	MAX	MIN	MAX		
<b>TIMING REQUIREMENTS</b>							
1	System Clock (CLK) Period	25	-	20	-	ns	
2	System Clock (CLK) LOW Time	6	-	5	-	ns	At 1.0V
3	System Clock (CLK) HIGH Time	9	-	7	-	ns	At 3.6V
17	System Clock (CLK) RISE Time	-	4	-	4	ns	1.0V to 3.6V
18	System Clock (CLK) FALL Time	-	4	-	4	ns	3.6V to 1.0V
4	Asynchronous Inputs SETUP Time	4	-	4	-	ns	(Note 37)
5	Asynchronous Inputs HOLD Time	4	-	4	-	ns	(Note 37)
6	RESET SETUP Time	10	-	10	-	ns	
7	RESET HOLD Time	0	-	0	-	ns	
8	Read Data SETUP Time	3	-	3	-	ns	
9	Read Data HOLD Time	2	-	2	-	ns	
10	$\overline{READY}$ SETUP Time	10	-	9	-	ns	
11	$\overline{READY}$ HOLD Time	3	-	3	-	ns	
20	Input RISE/FALL Times	-	6	-	6	ns	0.8V to 2.0V
<b>TIMING RESPONSES</b>							
12A	Status/ $\overline{PEACK}$ Active Delay	1	15	1	12	ns	1, (Notes 39, 42)
12B	Status/ $\overline{PEACK}$ Inactive Delay	1	16	1	13	ns	1, (Notes 39, 42)
13	Address Valid Delay	1	23	1	20	ns	1, (Notes 38, 39)
14	Write Data Valid Delay	0	27	0	24	ns	1, (Notes 38, 39)
15	Address/Status/Data Float Delay	0	25	0	24	ns	2, (Note 41)
16	HLDA Valid Delay	0	20	0	19	ns	1, (Notes 38, 39)
19	Address Valid to Status SETUP Time	9	-	12	-	ns	1, (Notes 39, 40)

**NOTES:**

- 37. Asynchronous inputs are INTR, NMI, HOLD, PREQ, ERROR, and BUSY. This specification is given only for testing purposes, to assure recognition at a specific CLK edge.
- 38. Delay from 1.0V on the CLK to 1.5V.
- 39. Output load:  $C_L = 100\text{pF}$ .
- 40. Delay measured from address reaching 1.5V to status reaching 1.5V.
- 41. Delay from 1.0V on the CLK to Float (no current drive) condition.
- 42. Delay from 1.0V on the CLK to 1.5V.

**AC Test Conditions**

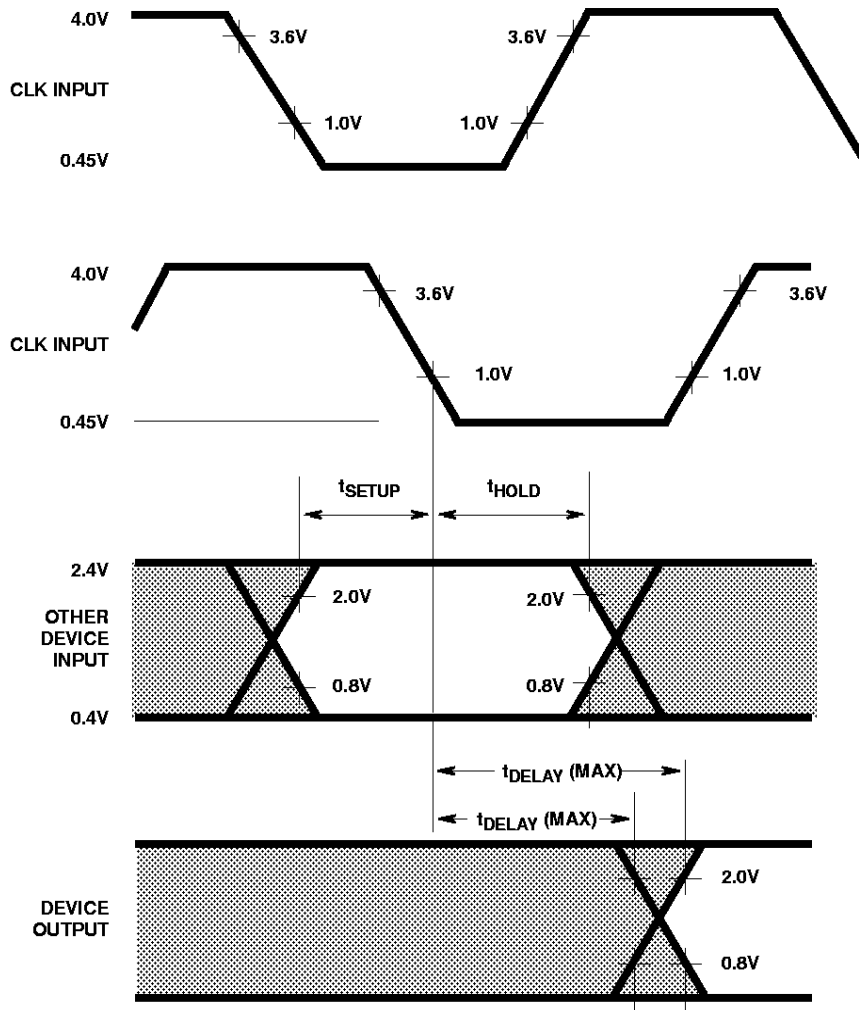
TEST CONDITION	$I_L$ (CONSTANT CURRENT SOURCE)	$C_L$
1	2.0mA	100pF
2	-6mA ( $V_{OH}$ to Float) 8mA ( $V_{OL}$ to Float)	100pF



# 80C286

## AC Specifications (Continued)

C80C86-12, -16  
 180C286-10, -12, -16  
 AC DRIVE AND MEASURE POINTS - CLK INPUT

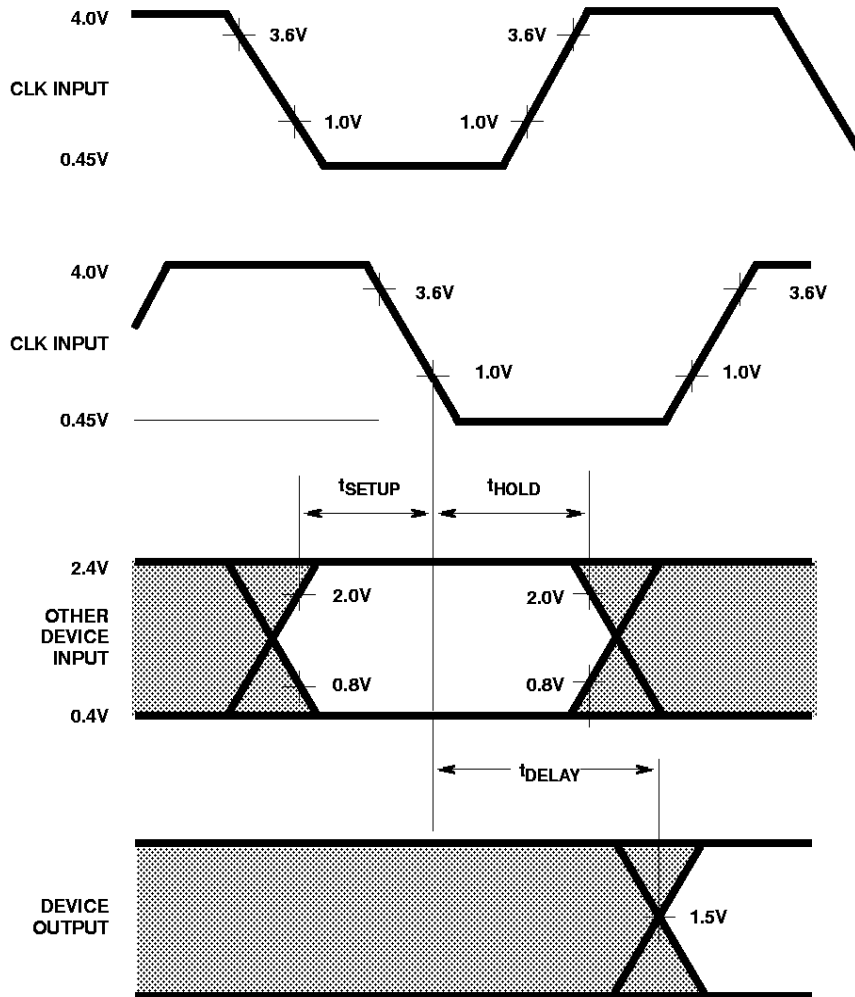


NOTE: For AC testing, input rise and fall times are driven at 1ns per volt.

FIGURE 32.

**AC Specifications** (Continued)

C80C286-20, -25  
I80C286-20  
AC DRIVE AND MEASURE POINTS - CLK INPUT



NOTE: Typical Output Rise/Fall Time is 6ns. For AC testing, input rise and fall times are driven at 1ns per volt.

FIGURE 33.

## 80C286

**AC Electrical Specifications** 82C284 and 82C288 Timing Specifications are given for reference only and no guarantee is implied.

### 82C284 Timing

SYMBOL	PARAMETER	10MHz		12.5MHz		16MHz		UNIT	TEST CONDITION
		MIN	MAX	MIN	MAX	MIN	MAX		
<b>TIMING REQUIREMENTS</b>									
11	SRDY/SRDYEN Setup Time	15	-	15	-	10	-	ns	
12	SRDY/SRDYEN Hold Time	2	-	2	-	1	-	ns	
13	ARDY/ARDYEN Setup Time	5	-	5	-	3	-	ns	(Note 43)
14	ARDY/ARDYEN Hold Time	30	-	25	-	20	-	ns	(Note 43)
<b>TIMING RESPONSES</b>									
19	PCLK Delay	0	20	0	16	0	15	ns	C <sub>L</sub> = 75pF, I <sub>OL</sub> = 5mA, I <sub>OH</sub> = 1mA

NOTE:

43. These times are given for testing purposes to ensure a predetermined action.

### 82C288 Timing

SYMBOL	PARAMETER	10MHz		12.5MHz		16MHz		UNIT	TEST CONDITION
		MIN	MAX	MIN	MAX	MIN	MAX		
<b>TIMING REQUIREMENTS</b>									
12	CMDLY Setup Time	15	-	15	-	10	-	ns	
13	CMDLY Hold Time	1	-	1	-	0	-	ns	
<b>TIMING RESPONSES</b>									
16	ALE Active Delay	1	16	1	16	1	12	ns	
17	ALE Inactive Delay	-	19	-	19	-	15	ns	
19	DT/R Read Active Delay	-	23	-	23	-	18	ns	C <sub>L</sub> = 150pF
20	DEN Read Active Delay	-	21	-	21	-	16	ns	I <sub>OL</sub> = 16mA Max
21	DEN Read Inactive Delay	3	23	3	21	5	14	ns	I <sub>OL</sub> = 1mA Max
22	DT/R Read Inactive Delay	5	24	5	18	5	14	ns	
23	DEN Write Active Delay	-	23	-	23	-	17	ns	
24	DEN Write Inactive Delay	3	23	3	23	3	15	ns	
29	Command Active Delay from CLK	3	21	3	21	3	15	ns	C <sub>L</sub> = 300pF
30	Command Inactive Delay from CLK	3	20	3	20	3	15	ns	I <sub>OL</sub> = 32mA Max

NOTE:

44. These times are given for testing purposes to ensure a predetermined action.

Waveforms

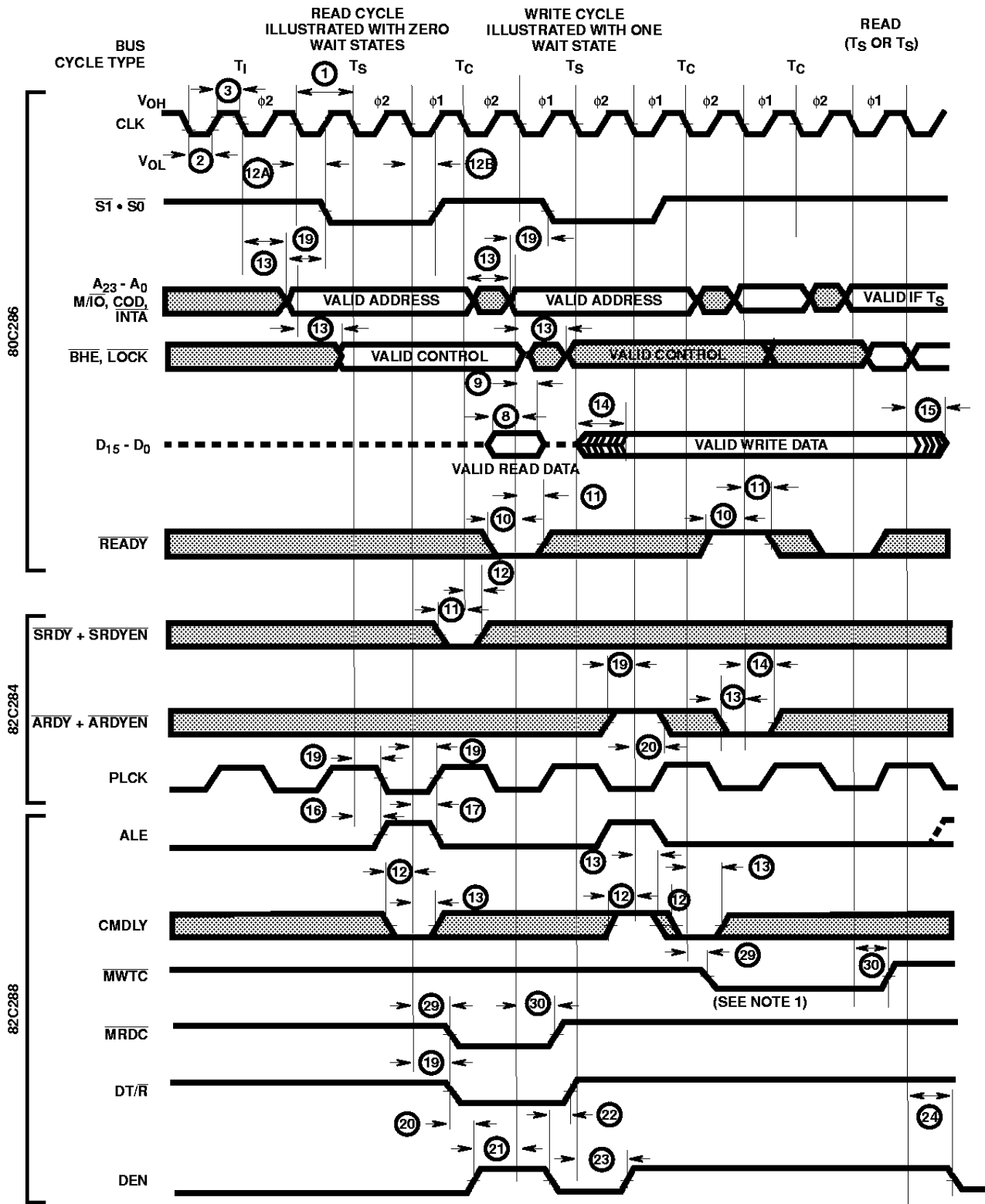


FIGURE 34. MAJOR CYCLE TIMING

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

Waveforms (Continued)

BUS CYCLE TYPE

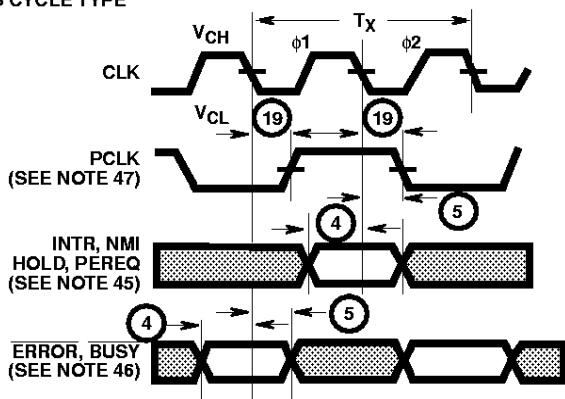


FIGURE 35. 80C286 ASYNCHRONOUS INPUT SIGNAL TIMING NOTES:

- 45. PCLK indicates which processor cycle phase will occur on the next CLK, PCLK may not indicate the correct phase until the first cycle is performed.
- 46. These inputs are asynchronous. The setup and hold times shown assure recognition for testing purposes.

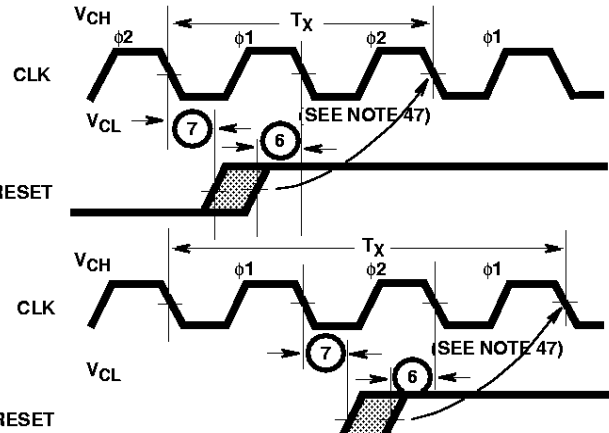


FIGURE 36. 80C286 RESET INPUT TIMING AND SUBSEQUENT PROCESSOR CYCLE PHASE

NOTE:

- 47. When RESET meets the setup time shown, the next CLK will start or repeat  $\phi_1$  of a processor cycle.

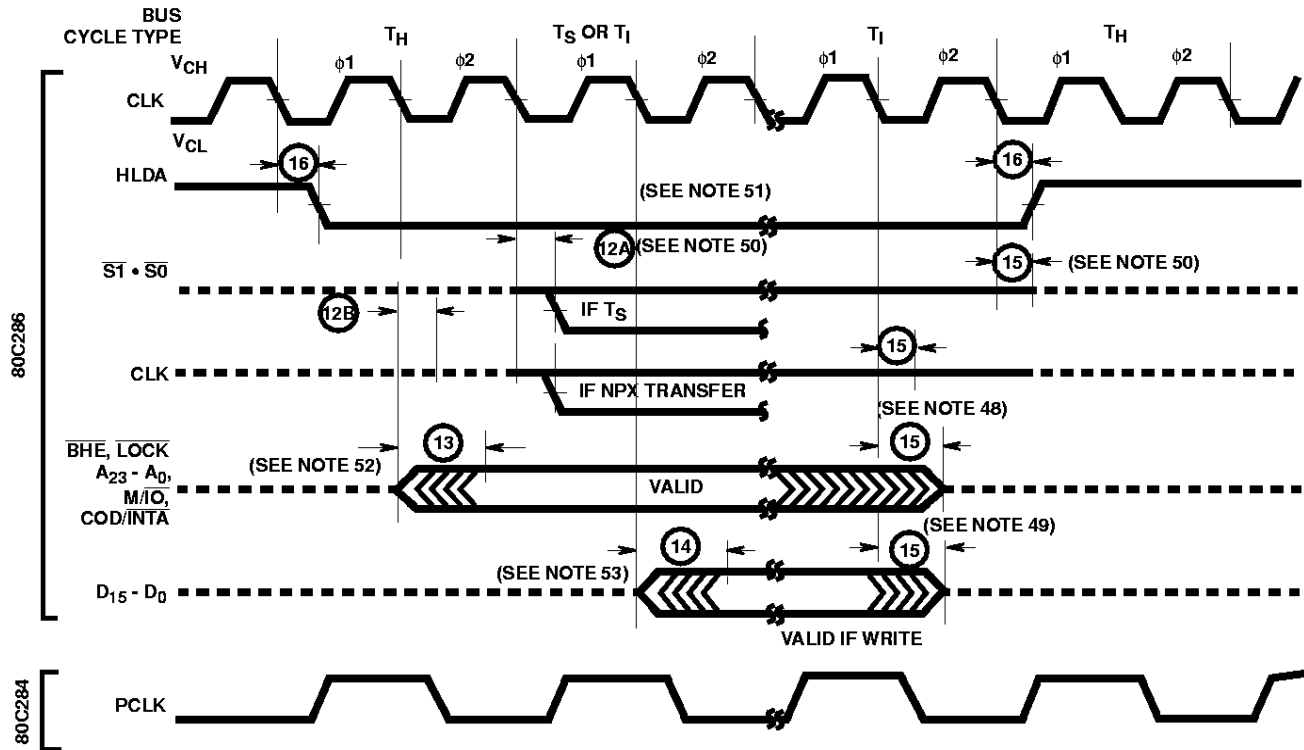
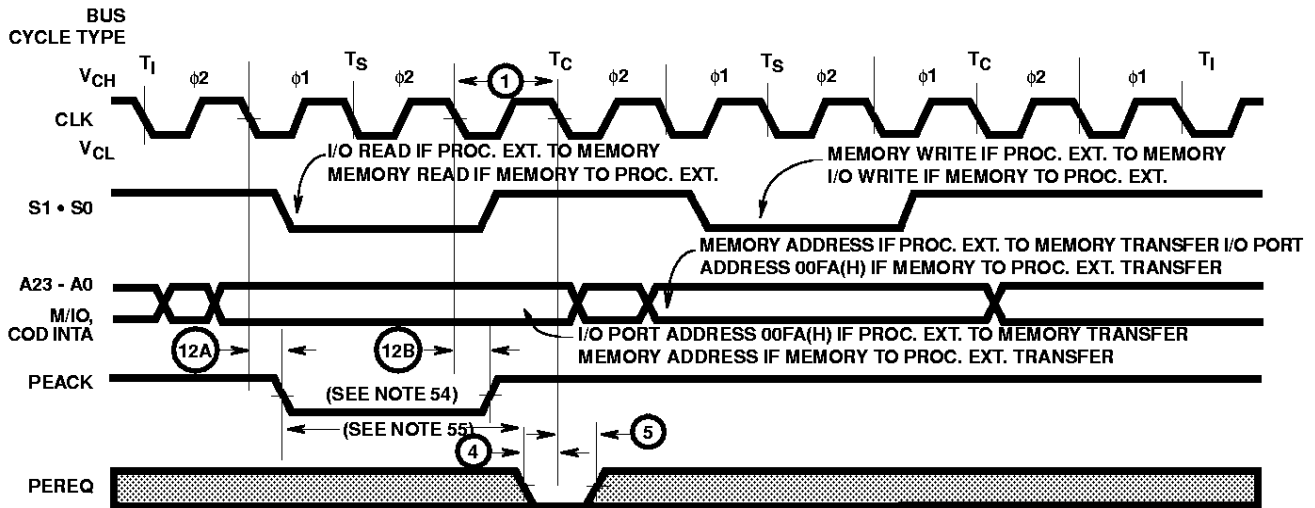


FIGURE 37. EXITING AND ENTERING HOLD

NOTES:

- 48. These signals may not be driven by the 80C286 during the time shown. The worst case in terms of latest float time is shown.
- 49. The data bus will be driven as shown if the cycle before  $T_I$  in the diagram was a write  $T_C$ .
- 50. The 80C286 puts its status pins in a high impedance logic one state during  $T_H$ .
- 51. For HOLD request set up to HLDA, refer to Figure 29.
- 52.  $\overline{BHE}$  and  $\overline{LOCK}$  are driven at this time but will not become valid until  $T_S$ .
- 53. The data bus will remain in a high impedance state if a read cycle is performed.

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/PEACK TIMING FOR ONE TRANSFER ONLY

NOTES:

- 54. 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  $3 \times \textcircled{1} - 12A_{MAX} - \textcircled{4}_{MIN}$ . The actual configuration dependent, maximum time is:  $3 \times \textcircled{1} - 12A_{MAX} - \textcircled{4}_{MIN} + N \times 2 \times \textcircled{1}$ . 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.

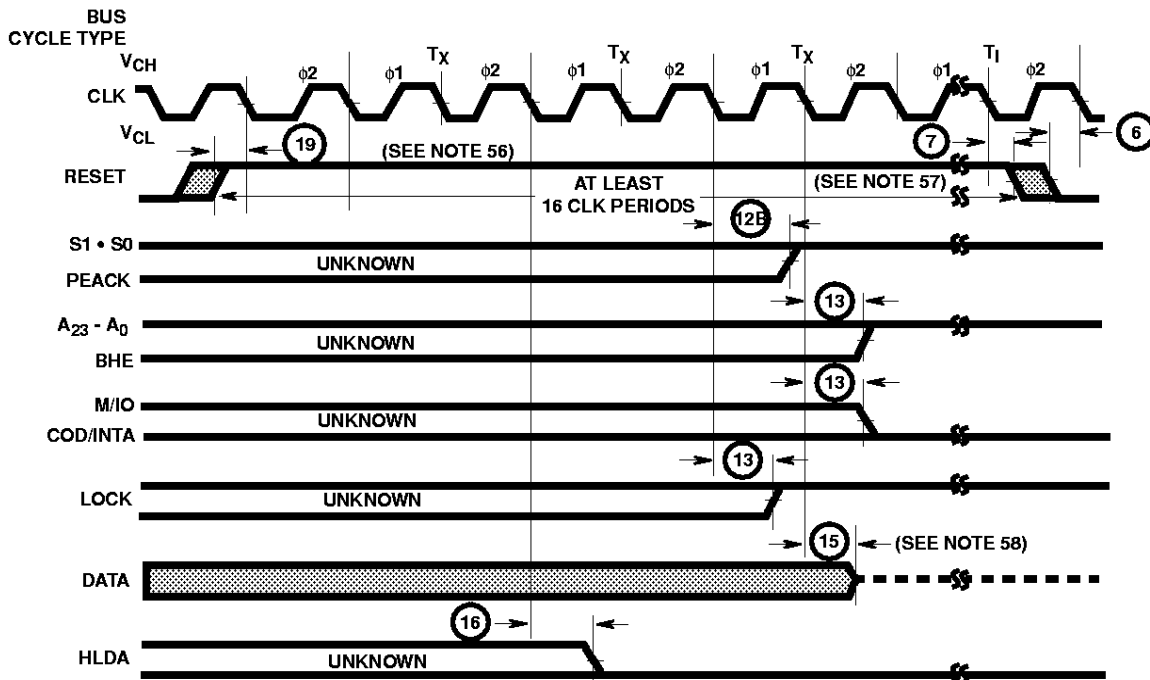


FIGURE 39. INITIAL 80C286 PIN STATE DURING RESET

NOTES:

- 56. Setup time for RESET ↑ may be violated with the consideration that φ1 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 φ1 or φ2.
- 58. The data bus is only guaranteed to be in a high impedance state at the time shown.

Waveforms (Continued)

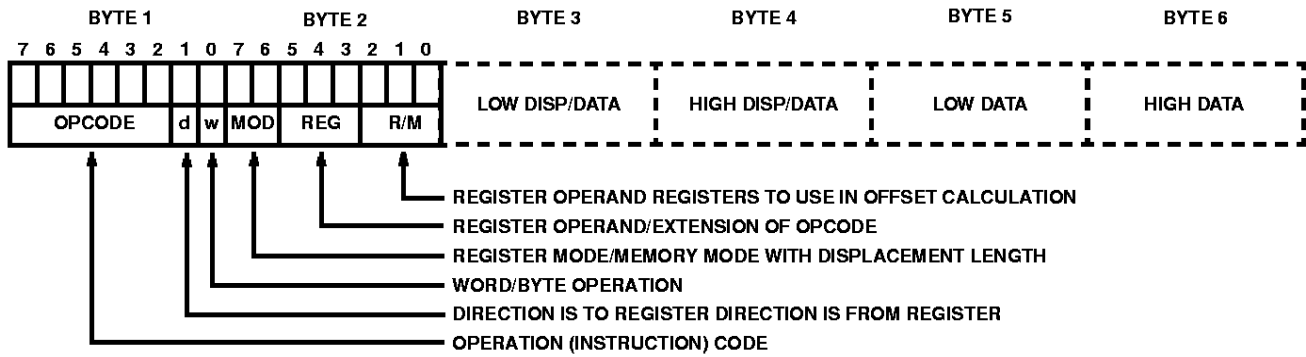


FIGURE 40A. SHORT OPCODE FORMAT EXAMPLE

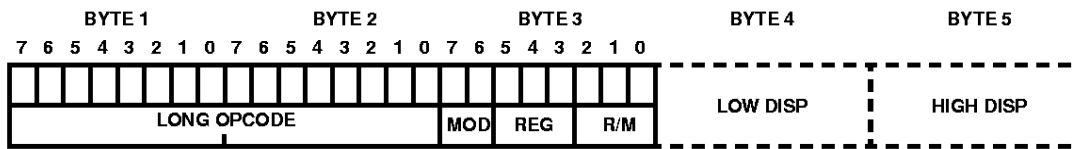


FIGURE 40B. LONG OPCODE FORMAT EXAMPLE

FIGURE 40. 80C286 INSTRUCTION FORMAT EXAMPLES

80C286 Instruction Set Summary

Instruction Timing Notes

The instruction clock counts listed below establish the maximum execution rate of the 80C286. With no delays in bus cycles, the actual clock count of an 80C286 program will average 5% more than the calculated clock count, due to instruction sequences which execute faster than they can be fetched from memory.

To calculate elapsed times for instruction sequences, multiply the sum of all instruction clock counts, as listed in the table below, by the processor clock period. An 12.5MHz processor clock has a clock period of 80 nanoseconds and requires an 80C286 system clock (CLK input) of 25MHz.

Instruction Clock Count Assumptions

1. The instruction has been perfected, decoded and is ready for execution. Control transfer instruction clock counts include all time required to fetch, decode, and prepare the next instruction for execution.
2. Bus cycles do not require wait states.
3. There are no processor extension data transfer or local bus HOLD requests.
4. No exceptions occur during instruction execution.

Instruction Set Summary Notes

Addressing displacements selected by the MOD field are not shown. If necessary they appear after the instruction fields shown.

Above/below refers to unsigned value.

Greater refers to more positive signed values.

Less refers to less positive (more negative) signed values

if d = 1, then "to" register; if d = 0 then "from" register

if w = 1, then word instruction; if w = 0, then byte instruction

if s = 0, then 16-bit immediate data form the operand

if s = 1, then an immediate data byte is sign-extended to form the 16-bit operand

x don't care

z used for string primitives for comparison with ZF FLAG

If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand

\* = add one clock if offset calculation requires summing 3 elements

n = number of times repeated

m = number of bytes of code in next instruction

Level (L) - Lexical nesting level of the procedure

The following comments describe possible exceptions, side effects and allowed usage for instructions in both operating modes of the 80C286.

**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.
6. A segment not-present violation occurs, a stack exception (12) occurs.
7. 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.
8. JMP, CALL, INT, RET, IRET instructions referring to another code segment will cause a general protection exception (13) if any privilege rule is violated.
9. A general protection exception (13) occurs if  $\text{CPL} \neq 0$ .
10. A general protection exception (13) occurs if  $\text{CPL} > \text{IOPL}$ .
11. The IF field of the flag word is not updated if  $\text{CPL} > \text{IOPL}$ . The IOPL field is updated only if  $\text{CPL} = 0$ .
12. 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.
13. 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.

**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
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
<b>DATA TRANSFER</b>								
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
Accumulator to Memory	1010001w	addr-low	addr-high		3	3	2	9



## 80C286

## 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/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
LEA = Load EA to Register	10001101	mod reg r/m			3 (Note 59)	3 (Note 59)		
LDS = Load Pointer to DS	11000101	mod reg r/m	(mod ≠ 11)		7 (Note 59)	21 (Note 59)	2	9, 10, 11
LES = Load Pointer to ES	11000100	mod reg r/m	(mod ≠ 1)		7 (Note 59)	21 (Note 59)	2	9, 10, 11
LAHF Load AH with Flags	10011111				2	2		

**80C286**

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

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
SAHF = Store AH into Flags	10011110				2	2		
PUSHF = Push Flags	10011100				3	3	2	9
POPF = Pop Flags	10011101				5	5	2, 4	9, 15
<b>ARITHMETIC</b>								
ADD = Add								
Reg/Memory with Register to Either	000000dw	mod reg r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9
Immediate to Register/Memory	100000sw	mod 000 r/m	data	data if sw = 01	3, 7 (Note 59)	3, 7 (Note 59)	2	9
Immediate to Accumulator	0000010w	data	data if w = 1		3	3		
ADC = Add with Carry								
Reg/Memory with Register to Either	000100dw	mod reg r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9
Immediate to Register/Memory	100000sw	mod 010 r/m	data	data if sw = 01	3, 7 (Note 59)	3, 7 (Note 59)	2	9
Immediate to Accumulator	0001010w	data	data if w = 1		3	3		
INC = Increment								
Register/Memory	1111111w	mod 000 r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9
Register	01000 reg				2	2		
SUB = Subtract								
Reg/Memory and Register to Either	001010dw	mod reg r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9
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		

## 80C286

### 80C286 Instruction Set Summary (Continued)

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
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 Adjust for Subtract	00101111				3	3		
MUL = Multiply (Unsigned)	1111011w	mod 100 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 = 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 = Integer 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

## 80C286 Instruction Set Summary (Continued)

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
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		
AAD = ASCII Adjust for Divide	11010101	00001010			14	14		
CBW = Convert Byte to Word	10011000				2	2		
CWD = Convert Word to Double Word	10011001				2	2		
<b>LOGIC</b>								
Shift/Rotate Instructions								
Register/Memory by 1	1101000w	mod TTT r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9
Register/Memory by CL	1101001w	mod TTT r/m			5+n, 8+n (Note 59)	5+n, 8+n (Note 59)	2	9
Register/Memory by Count	1100000	mod TTT r/m	count		5+n, 8+n (Note 59)	5+n, 8+n (Note 59)	2	9
			TTT	Instruction				
			000	ROL				
			001	ROR				
			010	RCL				
			011	RCR				
			100	SHL/SAL				
			101	SHR				
			111	SAR				
AND = And								
Reg/Memory and Register to Either	001000dw	mod reg r/m			2, 7 (Note 59)	2, 7 (Note 59)	2	9

80C286

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 to Register/Memory	1000000w	mod 100 r/m	data	data if w = 1	3, 7 (Note 59)	3, 7 (Note 59)	2	9
Immediate to Accumulator	0010010w	data	data if w = 1		3	3		
TEST = And Function to Flags, No Result								
Register/Memory and Register	1000010w	mod reg r/m			2, 6 (Note 59)	2, 6 (Note 59)	2	9
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
<b>STRING MANIPULATION</b>								
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
Repeated by Count in CX								
MOVS = Move String	11110011	1010010w			5 + 4n	5 + 4n	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
CMPS = Compare String	1111001z	1010011w			5 + 9n	5 + 9n	2, 8	8, 9
SCAS = Scan String	1111001z	1010111w			5 + 8n	5 + 8n	2, 8	8, 9
LODS = Load String	11110011	1010110w			5 + 4n	5 + 4n	2, 8	8, 9
STOS = Store String	11110011	1010101w			4 + 3n	4 + 3n	2, 8	8, 9
INS = Input String	11110011	0110110w			5 + 4n	5 + 4n	2	9, 14
OUTS = Output String	11110011	0110111w			5 + 4n	5 + 4n	2	9, 14
<b>CONTROL TRANSFER</b>								
CALL = Call								
Direct Within Segment	11101000	disp-low	disp-high		7 + m	7 + m	2	18
Register/Memory Indirect Within Segment	11111111	mod 010 r/m			7 + m, 11 + m (Note 59)	7 + m, 11 + m (Note 59)	2, 8	8, 9, 18
Direct Intersegment	10011010	Segment Offset			13 + m	26 + m	2	11, 12, 18
Protected Mode Only (Direct Intersegment)		Segment Selector						
Via Call Gate to Same Privilege Level						41 + m		8, 11, 12, 18
Via Call Gate to Different Privilege Level, No Parameters						82 + m		8, 11, 12, 18
Via Call Gate to Different Privilege Level, X Parameters						86 + 4x + m		8, 11, 12, 18
Via TSS						177 + m		8, 11, 12, 18
Via Task Gate						182 + m		8, 11, 12, 18
Indirect Intersegment	11111111	mod 011 r/m	mod ≠ 11		16 + m (Note 59)	29 + m (Note 59)	2	8, 9, 11, 12, 18
Protected Mode Only (Indirect Intersegment)								
Via Call Gate to Same Privilege Level						44 + m (Note 59)		8, 9, 11, 12, 18
Via Call Gate to Different Privilege Level, No Parameters						83 + m (Note 59)		8, 9, 11, 12, 18
Via Call Gate to Different Privilege Level, X Parameters						90 + 4x + m (Note 59)		8, 9, 11, 12, 18
Via TSS						180 + m (Note 59)		8, 9, 11, 12, 18
Protected Mode Only (Indirect Intersegment) (Continued)								
Via Task Gate						185 + m (Note 59)		8, 9, 11, 12, 18

## 80C286 Instruction Set Summary (Continued)

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
JMP = Unconditional Jump								
Short/Long	11101011	disp-low			7 + m	7 + m		18
Direct Within Segment	11101001	disp-low	disp-high		7 + m	7 + m		18
Register/Memory Indirect Within Segment	11111111	mod 100 r/m			7 + m, 11 + m (Note 59)	7 + m, 11 + m (Note 59)	2	9, 18
Direct Intersegment	11101010	Segment Offset			11 + m	23 + m		11, 12, 18
Protected Mode Only (Direct Intersegment)		Segment Selector						
Via Call Gate to Same Privilege Level						38 + m		8, 11,12,18
Via TSS						175 + m		8, 11,12,18
Via Task Gate						180 + m		8, 11,12,18
Indirect Intersegment	11111111	mod 101 r/m	mod ≠ 11		15 + m (Note 59)	26 + m (Note 59)	2	8, 9, 11, 12, 18
Protected Mode Only (Indirect Intersegment)								
Via Call Gate to Same Privilege Level						41 + m (Note 59)		8, 9, 11, 12, 18
Via TSS						178 + m (Note 59)		8, 9, 11, 12, 18
Via Task Gate						183 + m (Note 59)		8, 9, 11, 12, 18
RET = Return from CALL								
Within Segment	11000011				11 + m	11 + m	2	8, 9, 18
Within Segment Adding Immediate to SP	11000010	data-low	data-high		11 + m	11 + m	2	8, 9, 18
Intersegment	11001011				15 + m	25 + m	2	8, 9, 11, 12, 18
Intersegment Adding Immediate to SP	11001010	data-low	data-high		15 + m		2	8, 9, 11, 12, 18
Protected Mode Only (RET)								
To Different Privilege Level						55 + m		9, 11, 12, 18
JE/JZ = Jump on Equal Zero	01110100	disp			7 + m or 3	7 + m or 3		18
JL/JNGE = Jump on Less/Not Greater or Equal	01111100	disp			7 + m or 3	7 + m or 3		18
JLE/JNG = Jump on Less or Equal/not Greater	01111110	disp			7 + m or 3	7 + m or 3		18

## 80C286

## 80C286 Instruction Set Summary (Continued)

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
JB/JNAE = Jump on Below/Not Above or Equal	01110010	disp			7 + m or 3	7 + m or 3		18
JBE/JNA = Jump on Below or Equal/Not Above	01110110	disp			7 + m or 3	7 + m or 3		18
JP/JPE = Jump on Parity/Parity Even	01111010	disp			7 + m or 3	7 + m or 3		18
JO = Jump on Overflow	01110000	disp			7 + m or 3	7 + m or 3		18
JS = Jump on Sign	01111000	disp			7 + m or 3	7 + m or 3		18
JNE/JNZ = Jump on Not Equal/Not Zero	01110101	disp			7 + m or 3	7 + m or 3		18
JNL/JGE = Jump on Not Less/Greater or Equal	01111101	disp			7 + m or 3	7 + m or 3		18
JNLE/JG = Jump on Not Less or Equal/Greater	01111111	disp			7 + m or 3	7 + m or 3		18
JNB/JAE = Jump on Not Below/Above or Equal	01110011	disp			7 + m or 3	7 + m or 3		18
JNBE/JA = Jump on Not Below or Equal/Above	01110111	disp			7 + m or 3	7 + m or 3		18
JNP/JPO = Jump on Not Par/Par Odd	01111011	disp			7 + m or 3	7 + m or 3		18
JNO = Jump on Not Overflow	01110001	disp			7 + m or 3	7 + m or 3		18
JNS = Jump on Not Sign	01111001	disp			7 + m or 3	7 + m or 3		18
LOOP Loop CX Times	11100010	disp			8 + m or 4	8 + m or 4		18
LOOPZ/LOOPE = Loop While Zero/Equal	11100001	disp			8 + m or 4	8 + m or 4		18
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	11100000	disp			8 + m or 4	8 + m or 4		18
JCXZ = Jump on CX Zero	11100011	disp			8 + m or 4	8 + m or 4		18
ENTER = Enter Procedure	11001000	data-low	data-high	L			2, 8	8, 9
L = 0					11	11	2, 8	8, 9
L = 1					15	15	2, 8	8, 9
L > 1					16 + 4(L - 1)	16 + 4(L - 1)	2, 8	8, 9
LEAVE = Leave Procedure	11001001				5	5		
INT = Interrupt								
Type Specified	11001101	type			23 + m		2, 7, 8	
Type 3	11001100				23 + m		2, 7, 8	



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

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
INTO = Interrupt on Overflow	11001110				24 + m or 3 (3 if no interrupt)	(3 if no interrupt)	2, 6, 8	
Protected Mode Only								
Via Interrupt or Trap Gate to Same Privilege Level						40 + m		7, 8, 11, 12, 18
Via Interrupt or Trap Gate to Fit Different Privilege Level						78 + m		7, 8, 11, 12, 18
Via Task Gate						167 + m		7, 8, 11, 12, 18
IRET Interrupt Return	11001111				17 + m	31 + m	2, 4	8, 9, 11, 12, 15, 18
Protected Mode Only								
To Different Privilege Level						55 + m		8, 9, 11, 12, 15, 18
To Different Task (NT = 1)						169 + m		8, 9, 11, 12, 18
BOUND = Detect Value Out of Range	01100010	mod reg r/m			13 (Note 59)	13 (Use INT clock count if exception 5) (Note 59)	2, 6	6, 8, 9, 11, 12, 18
<b>PROCESSOR CONTROL</b>								
CLC = Clear Carry	11111000				2	2		
CMC = Complement Carry	11110101				2	2		
STC = Set Carry	11111001				2	2		
CLD = Clear Direction	11111100				2	2		
STD = Set Direction	11111101				2	2		
CLI = Clear Interrupt	11111010				3	3		14
STI = Set Interrupt	11111011				2	2		14
HLT = Halt	11110100				2	2		13
WAIT = Wait	10011011				3	3		
LOCK = Bus Lock Prefix	11110000				0	0		14
CTS = Clear Task Switched Flag	00001111	00000110			2	2	3	13

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

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE	REAL ADDRESS MODE	PROTECTED VIRTUAL ADDRESS MODE
ESC = Processor Extension Escape	11011TTT	mod LLL r/m			9-20 (Note 59)	9-20 (Note 59)	5, 8	8, 17
	(TTT LLL Are Opcode to Processor Extension)							
SEG = Segment Override Prefix	001 reg 110				0	0		
<b>PROTECTION CONTROL</b>								
LGDT = Load Global Descriptor Table Register	00001111	00000001	mod 010 r/m		11 (Note 59)	11 (Note 59)	2, 3	9, 13
SGDT = Store Global Descriptor Table Register	00001111	00000001	mod 000 r/m		11 (Note 59)	11 (Note 59)	2, 3	9
LIDT = Load Interrupt Descriptor Table Register	00001111	00000001	mod 011 r/m		12 (Note 59)	12 (Note 59)	2, 3	9, 13
SIDT = Store Interrupt Descriptor Table Register	00001111	00000001	mod 001 r/m		12 (Note 59)	12 (Note 59)	2, 3	9
LLDT = Load Local Descriptor Table Register From Register Memory	00001111	00000000	mod 010 r/m			17, 19 (Note 59)	1	9, 11, 13
SLDT = Store Local Descriptor Table Register To Register/Memory	00001111	00000000	mod 000 r/m			2, 3 (Note 59)	1	9
LTR = LTR = Local Task Register From Register/Memory	00001111	00000000	mod 011 r/m			17, 19 (Note 59)	1	9, 11, 13
STR = Store Task Register To Register Memory	00001111	00000000	mod 001 r/m			2, 3 (Note 59)	1	9
LMSW = Load Machine Status Word From Register/Memory	00001111	00000001	mod 110 r/m		3, 6 (Note 59)	3, 6 (Note 59)	2, 3	9, 13
SMSW = Store Machine Status Word	00001111	00000001	mod 100 r/m		2, 3 (Note 59)	2, 3 (Note 59)	2, 3	9
LAR = Load Access Rights From Register/Memory	00001111	00000010	mod reg r/m			14, 16 (Note 59)	1	9, 11, 16
LSL = Load Segment Limit From Register/Memory	00001111	00000011	mod reg r/m			14, 16 (Note 59)	1	9, 11, 16
ARPL = Adjust Requested Privilege Level: From Register/Memory		01100011	mod reg r/m		10, 11 (Note 59)	2	8, 9	
VERR = Verify Read Access: Register/Memory	00001111	00000000	mod 100 r/m		14, 16 (Note 59)	1	9, 11, 16	
VERR = Verify Write Access:	00001111	00000000	mod 101 r/m		14, 16 (Note 59)	1	9, 11, 16	

NOTE:

59. Shaded areas indicate instructions not available in 80G86/88 microsystems.

80C286 Machine Instruction Encoding Matrix

HI	LO															F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	
0	ADD b, f, r/m	ADD w, f, r/m	ADD b, t, r/m	ADD w, t, r/m	ADD b, ia	ADD w, ia	PUSH ES	POP ES	OR b, f, r/m	OR w, f, r/m	OR b, t, r/m	OR w, t, r/m	OR b, i	OR w, i	PUSH CS	PVAM n
1	ADC b, f, r/m	ADC w, f, r/m	ADC b, t, r/m	ADC w, t, r/m	ADC b, ia	ADC w, ia	PUSH SS	POP SS	SBB b, f, r/m	SBB w, f, r/m	SBB b, t, r/m	SBB w, t, r/m	SBB b, i	SBB w, i	PUSH DS	POP DS
2	AND b, f, r/m	AND w, f, r/m	AND b, t, r/m	AND w, t, r/m	AND b, ia	AND w, ia	SEG =ES	DAA	SUB b, f, r/m	SUB w, f, r/m	SUB b, t, r/m	SUB w, t, r/m	SUB b, i	SUB w, i	SEG =CS	DAS
3	XOR b, f, r/m	XOR w, f, r/m	XOR b, t, r/m	XOR w, t, r/m	XOR b, ia	XOR w, ia	SEG =SS	AAA	CMP b, f, r/m	CMP w, f, r/m	CMP b, t, r/m	CMP w, t, r/m	CMP b, i	CMP w, i	SEG =DS	AAS
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH SI	PUSH SI	PUSH DI	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6	PUSHA	POPA	BOUND	ARPL					PUSH w, i	IMUL w, t, r/m, i	PUSH b, i	IMUL b, t, r/m, i	INSB	INSW	OUTSB	OUTSW
7	JO	JNO	JB/ JNAE	JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA	JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNGE	JNL/ JGE	JLE/ JNG	JNLE/ JG
8	Immed b, r/m	Immed w, r/m	Immed b, r/m	Immed is, r/m	TEST b, r/m	TEST w, r/m	XCHG b, r/m	XCHG w, r/m	MOV b, f, r/m	MOV w, f, r/m	MOV b, t, r/m	MOV w, t, r/m	MOV sr, f, r/m	LEA	MOV sr, t, r/m	POP r/m
9	XCHG AX	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI	CBW	CWD	CALL i, d	WAIT	PUSHF	POPF	SAHF	LAHF
A	MOV m-AL	MOV m-AX	MOV AL-m	MOV AX-m	MOVSB	MOVSW	CMPSB	CMPSW	TEST b, i, a	TEST w, i, a	STOSB	STOSW	LODSB	LODSW	SCASB	SCASW
B	MOV i-AL	MOV i-CL	MOV i-DL	MOV i-BL	MOV i-AH	MOV i-CH	MOV i-DH	MOV i-BH	MOV i-AX	MOV i-CX	MOV i-DX	MOV i-BX	MOV i-SP	MOV i-BP	MOV i-SI	MOV i-DI
C	Shift b, i	Shift w, i	RET (i+SP)	RET	LES	LDS	MOV b, i, r/m	MOV w, i, r/m	ENTER	LEAVE	RET i, (i+SP)	RET i	INT Type 3	INT (any)	INTO	IRET
D	Shift b	Shift w	Shift b, CL	AAM	AAD		XLAT	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 6	ESC 7
E	LOOPNZ/ LOOPNE	LOOPZ/ LOOPE	LOOP	JCZ	IN b	IN w	OUT b	OUT w	CALL d	JMP d	JMP i, d	JMP si, d	IN DX, b	IN DX, w	OUT DX, b	OUT DX, w
F	LOCK		REP	REPZ	HLT	CMC	Grp 1 b, r/m	Grp 1 w, r/m	CLC	STC	CLI	STI	CLD	STD	Grp 2 b, r/m	Grp 2 w, r/m

where:

**80C286 Machine Instruction Encoding Matrix** (Continued)

mod r/mm	000	001	010	011	100	101	110	111
Immed	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
Shift	ROL	ROR	RCL	RCR	SHL/SAL	SHR	-	SAR
Grp 1	TEST	-	NOT	NEG	MUL	IMUL	DIV	IDIV
Grp 2	INC	DEC	CALL id	CALL l, id	JMP id	JMP l, id	PUSH	-
PVAM 0	SLDT	STR	LLDT	LTR	VERR	VERW	-	-
PVAM 1	SGDT	SIDT	LGDT	LIDT	SMSW	-	LMSW	-
PVAM 2	LAR							
PVAM 3	LSL							
PVAM 6	CLTS							

b = byte operation  
 d = direct  
 f = from CPU reg  
 i = immediate  
 ia = immediate to AX  
 id = indirect  
 is = immediate byte sign extension  
 l = long i.e., intersegment  
 n = 2nd byte of PVAM instruction  
 m = memory  
 r/m = EA is second byte  
 is = short intrasegment  
 sr = segment register  
 t = to CPU register  
 v = variable  
 w = word operation  
 z = zero

**Footnotes**

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

if mod = 11 then r/m is treated as a REG field

if mod = 00 then DISP = 0†, disp-low and disp-high are absent

if mod = 01 then DISP = disp-low sign extended to 16 bits, disp-high is absent

if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP

if r/m = 001 then EA = (BX) + (DI) + DISP

if r/m = 010 then EA = (BP) + (SI) + DISP

if r/m = 011 then EA = (BP) + (DI) + DISP

if r/m = 100 then EA = (SI) + DISP

if r/m = 101 then EA = (DI) + DISP

if r/m = 110 then EA = (BP) + DISP (Note 60)

if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data is required)

NOTE:

60. Except if mod = 00 and r/m = 110 then EQ = disp-high: disp-low.

**Segment Override Prefix**

0	0	1	reg	1	1	0
---	---	---	-----	---	---	---

reg is assigned according to the following:

REG	SEGMENT REGISTER
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

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

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