Preliminary

80C286

High-Speed CMOS 80286 Microprocessor

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- **Ultra high-performance processor**
  - —Over 20X performance of 8086
  - —Over 1.5X performance of 386SX-16*
  - —Comparable performance to 386 at same clock speed*
- **Wide range of clock rates**
  - —25 MHz (80C286-25)
  - —20 MHz (80C286-20)
  - —16 MHz (80C286-16)
  - —12.5 MHZ (80C286-12)
- **100% functionally and pin compatible with NMOS 286**
- **Static CMOS design for low power operation**
  - —Standby mode Icc = 5 mA maximum
  - —Operating mode Icc
    - 220 mA max at 12.5 MHz
    - 260 mA at max 16 MHz
    - 310 mA at max 20 MHz
    - 360 mA at max 25 MHz
- **68-lead LCC and 68-lead PLCC packages**

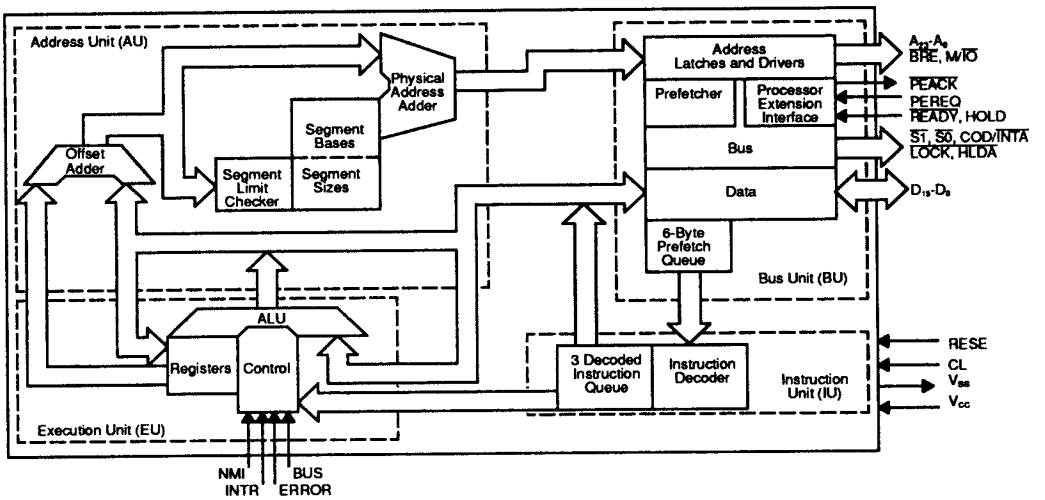
*When running 16-bit code (i.e., DOS or OS/2)

GENERAL DESCRIPTION

AMD's 80C286 is a high-speed implementation of the industry standard 80286 microprocessor. It is 100% functionally compatible with the NMOS version and is a plug compatible replacement. AMD's high-speed CMOS process allows clock speeds much higher than those attainable with NMOS. This CMOS 80286 operates at clock speeds up to 25 MHz.

This CMOS design is a static implementation which allows the processor to be clocked down to DC and still retain full register status. This is useful for designs where power consumption is a consideration as the 80C286 uses only 5 mA of supply current when in standby mode. The 80C286 also retains full functionality from it's maximum clock frequency through very low frequencies down to DC. Since power consumption is proportional to clock speed, the 80C286 may be clocked at a slower rate to draw less current.

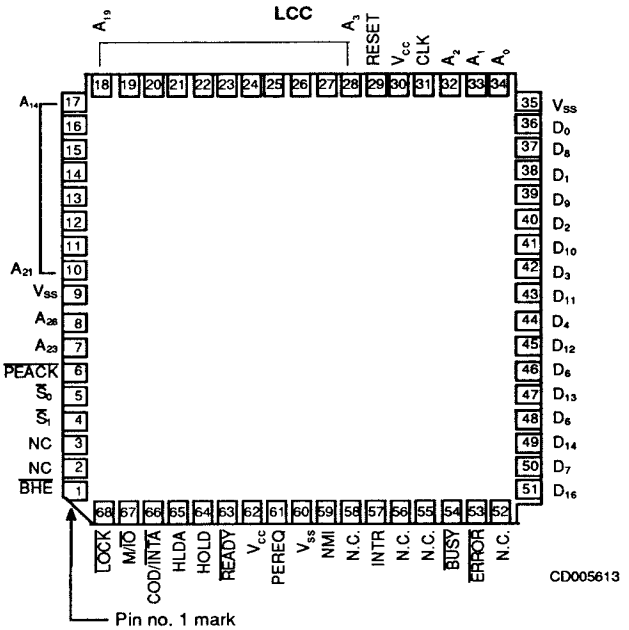
BLOCK DIAGRAM



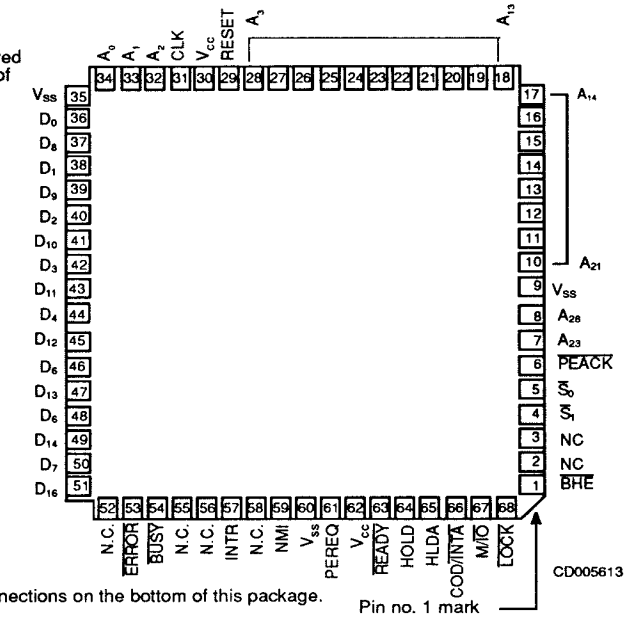
Publication # 09729 Rev. A Amendment /0
Issue Date: May 1989

CONNECTION DIAGRAMS

Component Pad Views—as viewed from underside of component on the PC board.



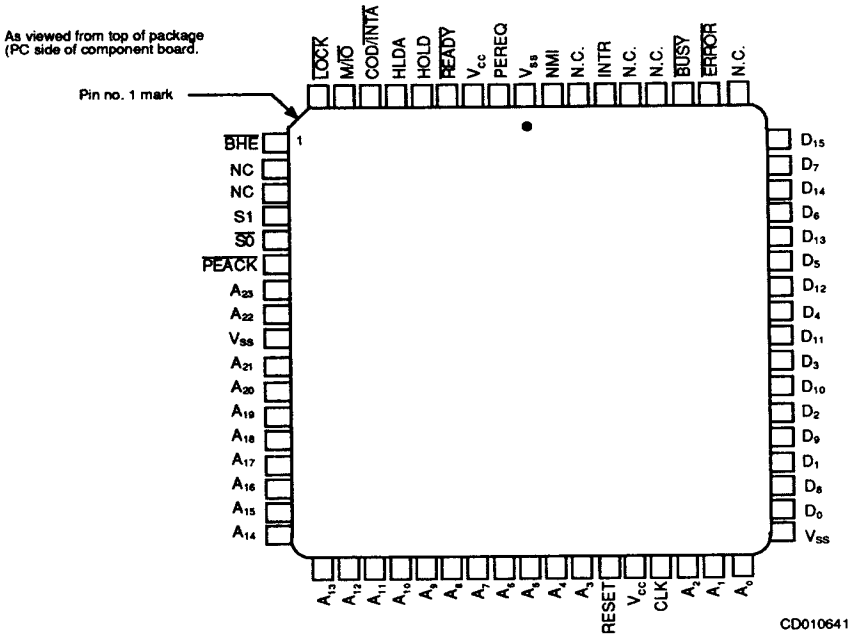
PC Board Views—as viewed from the component side of the PC board.



There are no electrical connections on the bottom of this package.

CONNECTION DIAGRAMS (continued)

PLCC



PIN DESIGNATIONS
(sorted by pin number)

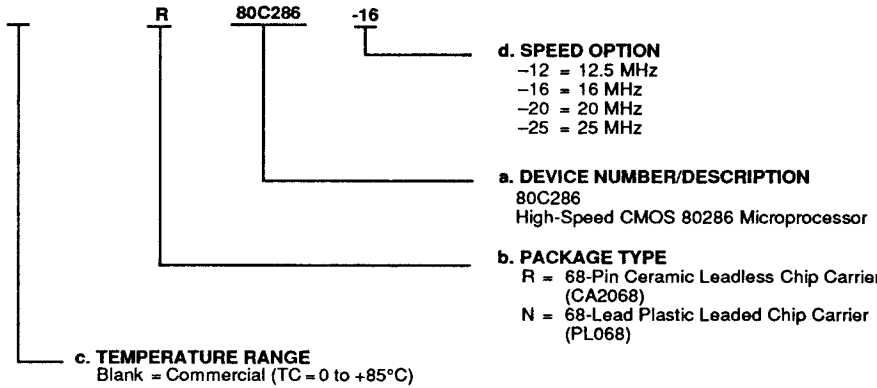
Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
1	$\overline{BHE}$	24	A_7	47	D_{13}
2	NC	25	A_6	48	D_6
3	NC	26	A_5	49	D_{14}
4	$\overline{S1}$	27	A_4	50	D_7
5	$\overline{S0}$	28	A_3	51	D_{15}
6	$\overline{PEACK}$	29	RESET	52	NC
7	A_{23}	30	V_{CC}	53	$\overline{ERROR}$
8	A_{22}	31	CLK	54	$\overline{BUSY}$
9	V_{SS}	32	A_2	55	NC
10	A_{21}	33	A_1	56	NC
11	A_{20}	34	A_0	57	INTR
12	A_{19}	35	V_{SS}	58	NC
13	A_{18}	36	D_0	59	NMI
14	A_{17}	37	D_8	60	V_{SS}
15	A_{16}	38	D_1	61	PEREQ
16	A_{15}	39	D_9	62	V_{CC}
17	A_{14}	40	D_2	63	$\overline{READY}$
18	A_{13}	41	D_{10}	64	HOLD
19	A_{12}	42	D_3	65	HLDA
20	A_{11}	43	D_{11}	66	COD/$\overline{INTA}$
21	A_{10}	44	D_4	67	M/$\overline{IO}$
22	A_9	45	D_{12}	68	$\overline{LOCK}$
23	A_8	46	D_5		

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The ordering number (Valid Combination) is formed by a combination of:

a. Device Number
b .Package Type
c. Temperature Range
d. Speed Option
e. Optional Processing



Valid Combinations	
80C286-25 80C286-20 80C286-16 80C286-12	R, N

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

PIN DESCRIPTION

CLK

System Clock (Input; Active HIGH)

System Clock provides the fundamental timing for 80C286 systems. 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 LOW-to-HIGH transition on the RESET input.

D_0-D_{15}

Data Bus (Input/Output; Active HIGH)

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 floats to three-state OFF during bus hold acknowledge.

A23-A0

Address Bus (Output; Active HIGH)

Address Bus outputs physical memory and I/O port addresses. A_0 is LOW when data is to be transferred on pins D_{7-0}. A_{23}-A_{16} are LOW during I/O transfers. The address bus is active HIGH and floats to three-state OFF during bus hold acknowledge.

$\overline{BHE}$

Bus High Enable (Output; Active LOW)

Bus High Enable indicates transfer of data on the upper byte of the data bus D_{15-8}. Eight-bit oriented devices assigned to the upper byte of the data bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is active LOW and floats to three-state OFF during bus hold acknowledge.

$\overline{BHE}$ and A_0 Encodings

$\overline{BHE}$ Value	A_0 Value	Function
0	0	Word transfer
0	1	Byte transfer on upper half of data bus (D_{15-8})
1	0	Byte transfer on lower half of data bus (D_{7-0})
1	1	Reserved

$\overline{S1}$, $\overline{S0}$

Bus Cycle Status (Output; Active LOW)

Bus Cycle Status indicates initiation of a bus cycle and, along with M/$\overline{IO}$ and COD/$\overline{INTA}$, defines the type of bus cycle. The bus is in a Ts state whenever one or both are LOW. $\overline{S1}$ and $\overline{S0}$ are active LOW and float to three-state OFF during bus hold acknowledge.

80C286 Bus Cycle Status Definition

COD/$\overline{INTA}$	M/$\overline{IO}$	$\overline{S1}$	$\overline{S0}$	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_1 = 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	0	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

M/$\overline{IO}$

Memory/IO Select (Output)

Memory/IO Select distinguishes memory access from I/O access. If HIGH during Ts, 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/$\overline{IO}$ floats to three-state OFF during bus hold acknowledge.

COD/$\overline{INTA}$

Code/Interrupt Acknowledge (Output)

Code/Interrupt Acknowledge distinguishes instruction fetch cycles from memory data read cycles. Also distinguishes interrupt acknowledge cycles from I/O cycles. COD/$\overline{INTA}$ floats to three-state OFF during bus hold acknowledge.

$\overline{LOCK}$

Bus Lock (Output; Active LOW)

Bus Lock indicates that other system bus masters are not to gain control of the system bus following the current bus cycle. 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 floats to three-state OFF during hold acknowledge.

PIN DESCRIPTION (continued)

READY
Bus Ready (Input; Active LOW)

Bus Ready terminates a bus cycle. Bus cycles are extended without limit until terminated by $\overline{\text{READY}}$ LOW. $\overline{\text{READY}}$ is an active LOW synchronous input requiring set-up and hold times relative to the system clock be met for correct operation. $\overline{\text{READY}}$ is ignored during bus hold acknowledge.

HOLD, HLDA
Bus Hold Request and Hold Acknowledge (Input/Output; Active HIGH)

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 to three-state OFF and then active 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 buys. This terminates the bus hold acknowledge condition. HOLD may be asynchronous to the system clock. These signals are active HIGH.

INTR
Interrupt Request (Input; Active HIGH)

Interrupt Request requests 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 assure program interruption, INTR must remain active until the first interrupt acknowledge cycle is completed. 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
Non-maskable Interrupt Request (Input; Active HIGH)

Non-maskable Interrupt Request interrupts the 80C286 with an internally supplied vector value of 2. 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, $\overline{\text{PEACK}}$
Processor Extension Operand Request and Acknowledge (Input/Output)

Processor Extension Operand Request and Acknowledge extends 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 $\overline{\text{PEACK}}$ output signals the processor extension when the requested operand is being transferred. PEREQ is active HIGH and may be asynchronous to the system clock. $\overline{\text{PEACK}}$ is active LOW.

$\overline{\text{BUSY}}$, $\overline{\text{ERROR}}$
Processor Extension Busy and Error

Processor Extension Busy and Error indicate the operating condition of a processor extension to the 80C286. An active $\overline{\text{BUSY}}$ input stops 80C286 program execution on WAIT and some ESC instructions until $\overline{\text{BUSY}}$ becomes inactive (HIGH). The 80C286 may be interrupted while waiting for $\overline{\text{BUSY}}$ to become inactive. An active $\overline{\text{ERROR}}$ 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.

RESET
System Reset (Input; Active HIGH)

System Reset clears the internal logic of the 80C286 and is active HIGH. The 80C286 may be reinitialized 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:

80C286 Pin State during Reset

Pin Value	Pin Names
1 (HIGH)	$\overline{S0}$, $\overline{ST}$, $\overline{\text{PEACK}}$, A_{23}-A_0, $\overline{BHE}$, $\overline{LOCK}$
0 (LOW)	M/$\overline{IO}$, COD/$\overline{INTA}$, HLDA
three-state OFF	D_{15}-D_0

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.

PIN DESCRIPTION (continued)

A LOW-to-HIGH transition of RESET synchronous to the system clock, will begin a new processor cycle at the next 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 period. Synchronous LOW-to-HIGH transitions of RESET are only required for systems where the processor clock must be phase synchronous to another clock.

V_{SS}
System Ground (Input; Active HIGH)

System Ground: 0 volts.

V_{CC}
System Power (Input; Active HIGH)

System Power: +5 volt power supply.

FUNCTIONAL DESCRIPTION

Introduction

The 80C286 is a fully static advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multitasking systems. Depending on the application, the 80C286's performance is up to 20 times faster than the standard 5-MHz 8086, while providing complete upward software compatibility with AMD's iAPX 86, 88, and 186 family of CPUs.

Static Operation

AMD's 80C286 is composed of complete static circuitry. Unlike the dynamic circuit design, the 80C286's internal registers, counters, and latches are static and do not require refresh which eliminates the minimum operating frequency restriction that is typically placed on microprocessors.

AMD's 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. Additionally, a significant decrease in power requirement occurs if the clock is stopped in phase two of the processor clock cycle. Details on clock relationships can be found in the Bus Operation section.

Note that the ability to stop the clock to processor is useful for system debug or power critical applications such as battery-powered laptop personal computers. The 80C286 can be single-stepped using only the CPU clock, and this state can be maintained as long as necessary. Single step clock operation allows for simple interface circuitry to provide critical information during system debug.

Static design 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 (5 mA maximum).

The 80C286 operates in two modes: iAPX 86 real address mode and protected virtual address mode. Both modes execute a superset of the iAPX 86 and 88 instruction set.

In iAPX 86 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 task's programs and data. Both modes provide the same base instruction set, registers, and addressing modes.

The following pages describe first, the base 80C286 architecture common to both modes; second, iAPX 86 real address mode; and third, protected mode.

80C286 Base Architecture

The iAPX 86, 88, 286, and C286 CPU family all contain the same basic set of registers, instructions, and addressing modes. The 80C286 processor is upward compatible with the 8086, 8088, and 80186 CPUs.

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 aspect of the 80C286 processor state. These include the Instruction Pointer, which contains the offset address of the next sequential instruction to be executed.

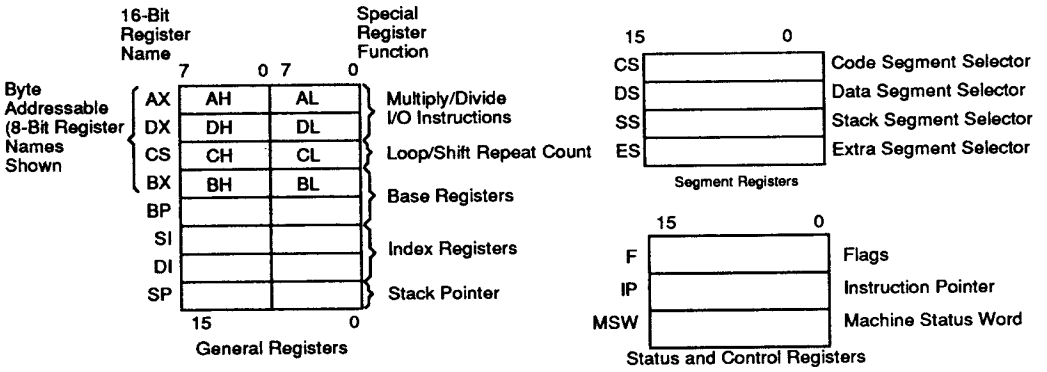


Figure 1. Register Set

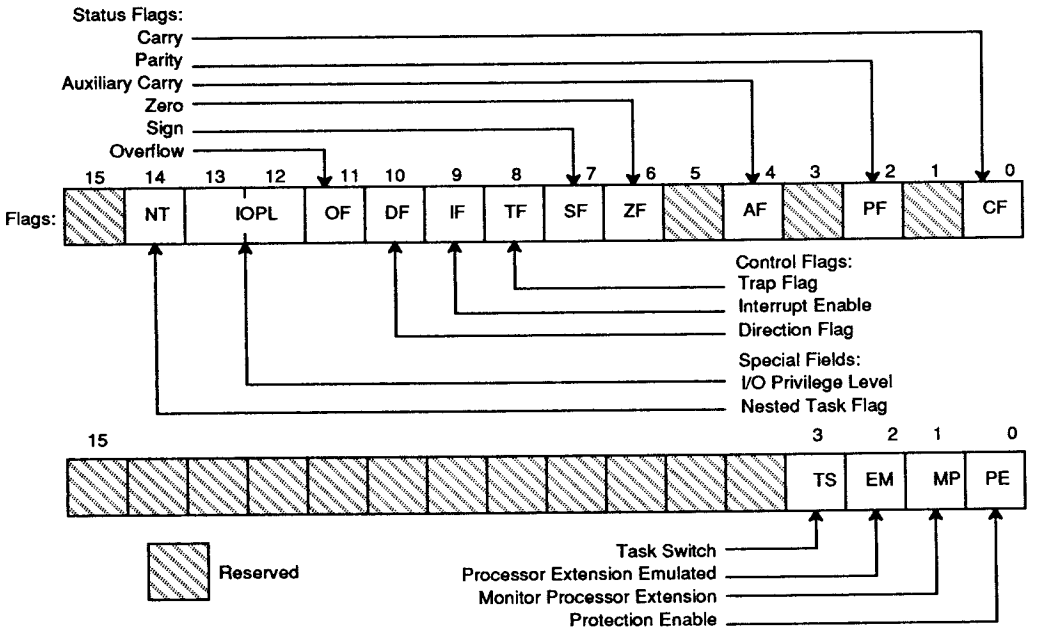


Figure 2. Status and Control Register Bit Functions

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.

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 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, shift/rotate/logical, string manipulation, program transfer, high-level instructions, and processor control. These categories are summarized in Figures 3–9.

An 80C286 instruction can reference zero, one, or two operands where an operand resides 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 Register
Immediate to Register
Memory to Memory
Register to Memory
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.

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.

Memory Organization

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 an offset to address a memory operand.

	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

Figure 3. Data Transfer 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
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
	Division
CBW	Convert byte to word
CWD	Convert word to doubleword

Figure 4. Arithmetic 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

Figure 5. String 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

Figure 6. Shift/Rotate/Logical 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		
JG/JNLE	Jump if greater/not less nor equal	LOOP	Loop
JGE/JNL	Jump if greater or equal/not less	LOOPE/LOOPZ	Loop if equal/zero
JL/JNGE	Jump if less/not greater nor equal	LOOPNE/LOOPNZ	Loop if not equal/not zero
JLE/JNG	Jump if less or equal/not greater	JCXZ	Jump if register CX = 0
JNC	Jump if not carry		
JNE/JNZ	Jump if not equal/not zero	**Interrupts**	
JNO	Jump if not overflow		
JNP/JPO	Jump if not parity/parity odd	INT	Interrupt
JNS	Jump if not sign	INTO	Interrupt if overflow
JO	Jump if overflow	IRET	Interrupt return
JP/JPE	Jump if parity/parity even		
JS	Jump if sign		

Figure 7. Program Transfer 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 BUSY not 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

Figure 8. Processor Control Instructions

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

Figure 9. High-Level Instructions

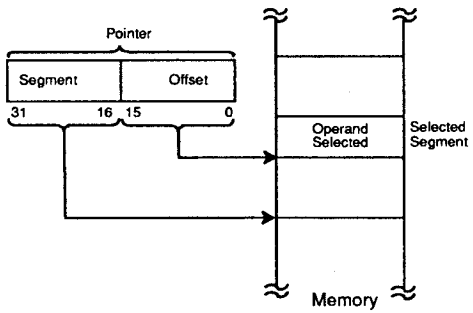


Figure 10. Two-Component Address

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

Most instructions need not explicitly specify which segment register is used. The correct segment register is automatically chosen according to the rules of Table 2. These rules follow the way programs are written (see Figure 11) as independent modules that require areas for code and data, a stack, and access to external data areas.

Memory Reference Needed	Segment Register Used	Selection Rule 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.

Table 2. Segment Register Selection Rules

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 that do not reside in one of the four immediately available segments, either a full 32-bit pointer can be used or a new segment selector must be loaded.

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)

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, here described.

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).

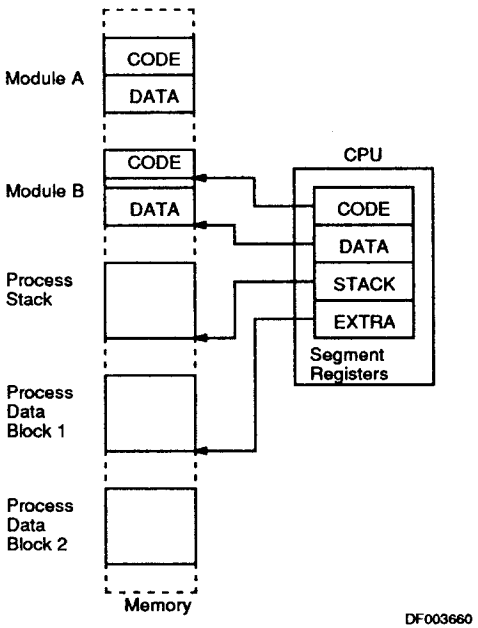


Figure 11. Segmented Memory Helps Structure Software

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.

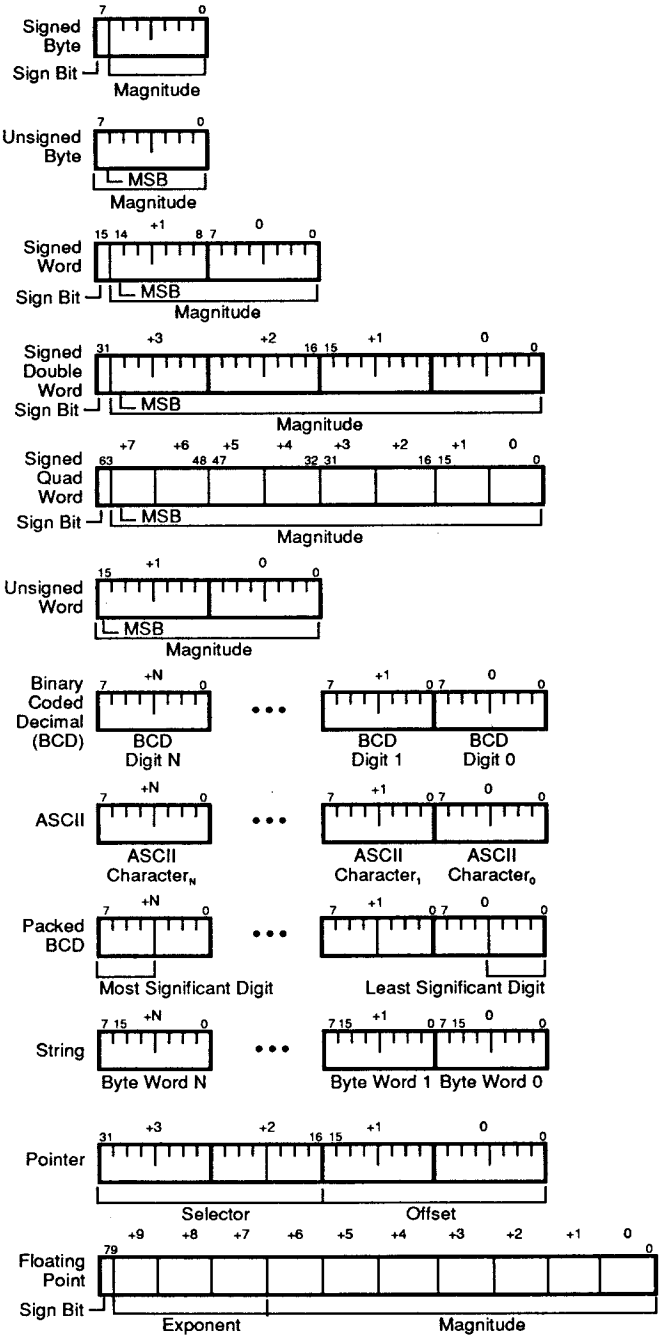
Based Indexed Mode with Displacement: The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

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 two's complement representation. signed 32- and 64-bit integers are supported using the 80C287 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 80C287 Numeric Processor configuration.)

Figure 12 graphically represents the data types supported by the 80C286.



*Supported by iAPX 80C286/80C287 Numeric Data Processor Configuration

DF003670

Figure 12. 80C286 Supported Data Types

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. 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. Eight-bit port addresses are zero extended such that A15-A8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Table 3. Interrupt Vector Assignments

Function	Interrupt Number	Related Instructions	Return Address Before Instruction Causing Exception?
Divide error exception	0	DIV, IDIV	Yes
Single step interrupt	1	All	
NMI interrupt	2	All	
Breakpoint interrupt	3	INT	
INTO detected overflow exception	4	INTO	No
BOUND range exceeded exception	5	BOUND	Yes
Invalid opcode exception	6	Any undefined opcode	Yes
Processor extension not available exception	7	ESC or WAIT	Yes
Reserved	8–15		
Processor extension error input	16	ESC or WAIT	
Reserved	17–31		
User defined	32–255		

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 at 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. Non-maskable 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.

Further maskable interrupts are disabled while servicing an interrupt by resetting the IF but 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 not service further NMI requests, INTR requests, or 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 4. Interrupt processing involves saving the flags, return address, and setting CS:IP to point at the first instruction of the interrupt handler. If other interrupts remain enabled, they are processed before the first instruction of the current interrupt handler is executed. The last interrupt processed is therefore the first one serviced.

Table 4. Interrupt Processing Order

Order	Interrupt
1	INT instruction or exception
2	Single step
3	NMI
4	Processor extension segment overrun
5	INTR

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 5.

Table 5. 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)

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 6, control the processor extension interface. After RESET, this register contains FFF0(H) which places the 80C286 in iAPX 86 real address mode.

Table 6. 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 7.

Table 7. Recommended MSW Encodings For Processor Extension Control

TS	MP	EM	Recommended Use	Instructions Causing Exception
0	0	0	iAPX 86 real address mode only. Initial encoding after RESET. 80C286 operation is identical to iAPX 86, 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 (number 7) on WAIT allows software to test for an error pending from a previous processor extension operation.	ESC or WAIT

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.

iAPX 80C286 Real Address Mode

The 80C286 executes a fully upward-compatible superset of the 8086 instruction set in real address mode. In real address mode the 80C286 is object code compatible with 8086 and 8088 software. The real address mode architecture (registers and addressing modes) is exactly as described in the 80C286 Base Architecture section.

Memory Size

Physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A0 through A19 and $\overline{BHE}$. A20 through A23 are ignored.

Memory Addressing

In real address mode the processor generates 20-bit physical addresses directly from a 20-bit segment base address and a 16-bit offset.

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 13 for a graphic representation of address formation.

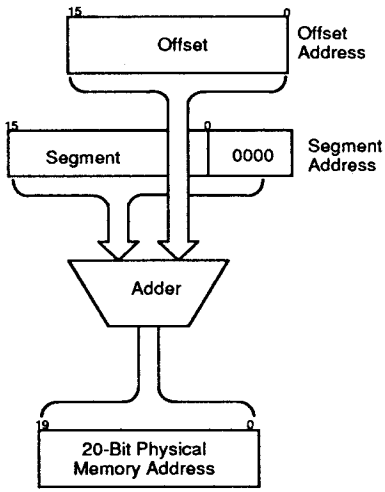


Figure 13. 80C286 Real Address Mode Address Calculation

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 overlayed by another segment to reduce physical memory requirements.

Reserved Memory Locations

The 80C286 reserves two fixed areas of memory in real address mode (see Figure 14): 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.

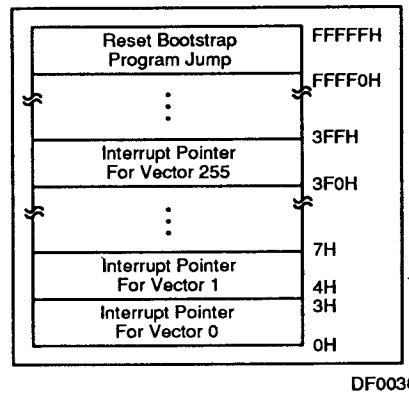


Figure 14. 80C286 Real Address Mode Initially Reserved Memory Locations

Table 8. Real Address Mode Addressing Interrupts

Function	Interrupt Number	Related Instructions	Return Address Before Instruction?
Interrupt table limit too small exception	8	INT vector 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

Interrupts

Table 8 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 000000(H) and its size set to 03FF(H). These values are compatible with iAPX 86, 88 software. LIDT should only be executed in preparation for the protected mode.

Shutdown

Shutdown occurs when a severe error is detected that prevents further instruction processing by the CPU. Shutdown and halt are externally signaled via a halt bus operation. They can be distinguished by A1 HIGH for halt and A1 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 POP 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 8086 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 remain the same. Programs for the iAPX 86, 88, 186, and real address model 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 pin A23–A0 and $\overline{BHE}$. 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.

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 table in memory. The 16-bit offset is added to the segment base address to form the physical address as shown in Figure 15. The tables are automatically referenced by the CPU whenever a segment register is loaded with a selector. All 80286 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.

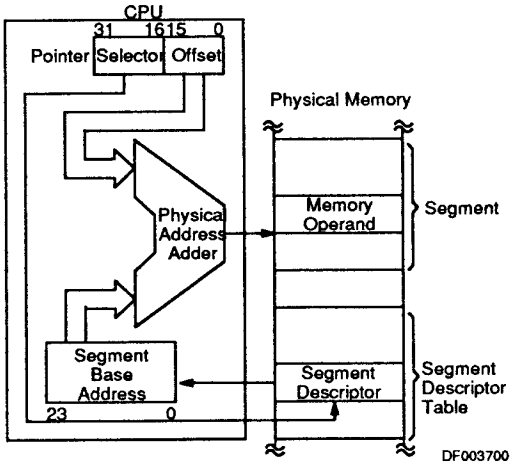


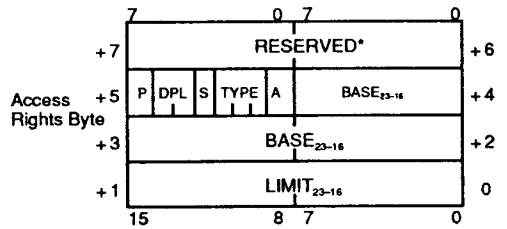
Figure 15. Protected Mode Memory Addressing

Descriptors

Descriptors defined 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 Figure 16). Any segment usage violating a segment attribute indicate by the segment descriptor will prevent the memory cycle and cause an exception or interrupt.



* Must be set to 0 for compatibility with iAPX 386 and future upgrades

DF003710

Access Rights Byte Definition

	Bit Position	Name		Function	
Type Field Definition	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. Segment privilege attribute used in privilege tests.	
	6–5	Descriptor Privilege Level (DPL)			
	4	Segment Descriptor (S)	S = 1	Code or Data Segment descriptor	
			S = 0	Non-segment descriptor	
	3	Executable (E)	E = 0	Data segment descriptor type is:	Data Segment
	2	Expansion Direction (ED)	ED = 0	Grow up segment, offsets must be < limit.	Data Segment
			ED = 1	Grow down segment, offsets must be > limit.	Data Segment
	1	Writable (W)	W = 0	Data segment may not be written into.	Data Segment
			W = 1	Data segment may be written into.	Data Segment
	3	Executable (E)	E = 1	Code Segment Descriptor type is:	Code Segment
	2	Conforming (C)	C = 1	Code segment may only be executed when CPL > DPL.	Code Segment
	1	Readable (R)	R = 0	Code segment may not be read.	Code Segment
			R = 1	Code segment may be read.	Code Segment
	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.	

Figure 16. Code and Data Segment Descriptors

Code and data are stored in two types of segments: code segments and data segments. Both types are identified and defined by segment descriptors. 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 descrip-

tor. DPL controls when the descriptor may be used by a task (refer to privilege discussion). 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. 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; upward (ED = 0) for data segments, and downward (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 Figure 16).

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).

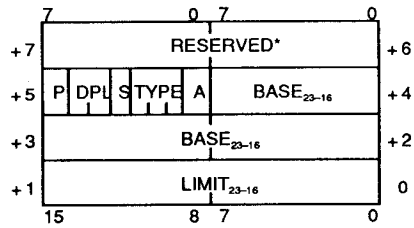
System Segment Descriptors

(S = 0, TYPE 1-3)

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

Figure 17 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 Figure 17.

System Segment Descriptor



* Must be set to 0 for compatibility with iAPX 386 and future upgrades

TB0000880

System Segment Descriptor Fields

Name	Value	Description
TYPE	1	Available Task State Segment
	2	Local Descriptor Table Descriptor
	3	Busy Task State Segment
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

Figure 17. System Segment Format

Gate Descriptors

(S = 0, TYPE = 4-7)

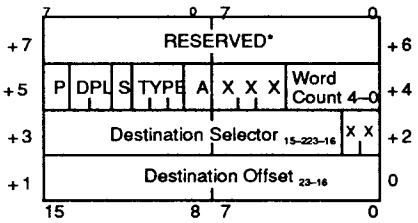
Gates are used to control access to entry points within the target code segment. The gate descriptors are call gates, task gates, interrupt gates and trap gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the CPU to automatically perform protection checks and control the 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 gates does not.

Figure 18 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.

Gate Descriptor



* Must be set to 0 for compatibility with iAPX 386 and future upgrades

TB0000880

Gate Descriptor Fields

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 Trap Gate) Selector to the target task state segment (Task Gate)
DESTINATION OFFSET	16-bit offset	Entry point within the target code segment

Figure 18. Gate Descriptor Format

The access byte format is the same for all gate 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). Bit 4 must equal 0 to indicate a system control descriptor. The Type field specifies the descriptor type as indicated in Figure 18.

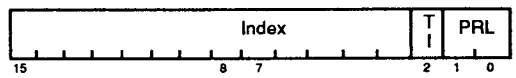
Segment Descriptor Cache Registers

A segment descriptor 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 20) 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 memory. 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.

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 19. 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).

Selector



Bits	Name	Function
1–0	Requested Privilege Level (RPL)	Indicates Selector Privilege Level Desired
2	Table Indicator (TI)	TI = 0 To use Global Descriptor Table (GDT) TI = 1 Use Local Descriptor Table (LDT)
15–3	Index	Select Descriptor Entry in Table

Figure 19. Selector Fields

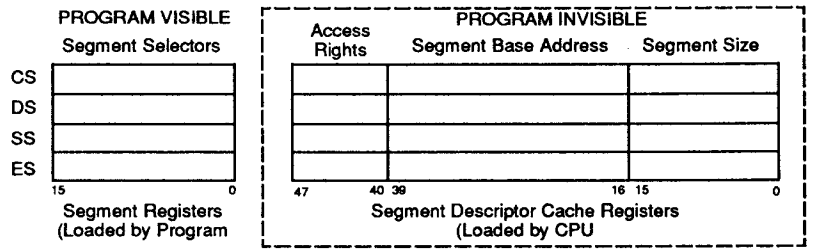


Figure 20. Descriptor Cache Registers

Local and Global Descriptor Tables

Two tables of 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 confines descriptor access to the defined limits of the table as shown in Figure 21. A restartable exception (13) will occur if an attempt is made to reference a descriptor outside the table limits.

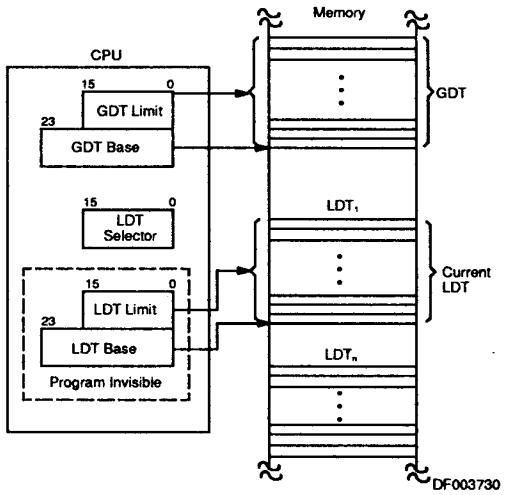


Figure 21. Local and Global Descriptor Table Definition

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. A segment cannot be accessed by a task if its segment descriptor does not exist in either descriptor table at the time of access.

The LGDT and LLDT instructions load the base and limit of the global and local descriptor tables. LGDT and LLDT are protected. 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 base address of the Global Descriptor Table as shown in Figure 22. The LLDT instruction loads a selector which refers to a descriptor in the Local Descriptor Table. This descriptor contains the base address and limit for an LDT, as shown in Figure 17.

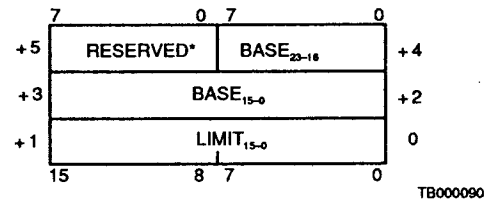


Figure 22. Global Descriptor Table and Interrupt Descriptor Data Type

* Must be set to 0 for compatibility with iAPX 386 and future upgrades.

Interrupt Descriptor Table

The protected mode 80C286 has a third descriptor table, called the Interrupt Descriptor Table (IDT) (see Figure 23), 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 base and 16-bit limit register in the CPU. The protected LIDT instruction loads these registers with a six-byte value of identical form to that of the LGDT instruction (see Figure 22 and Protected Mode Initialization).

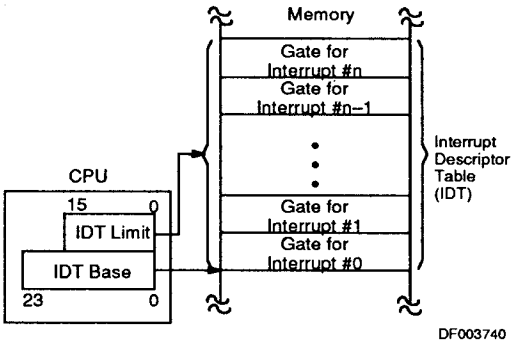


Figure 23. Local and Global Descriptor Table Definition

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.

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 24, is an extension of the user/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. Tasks may also have a separate stack for each privilege level.

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

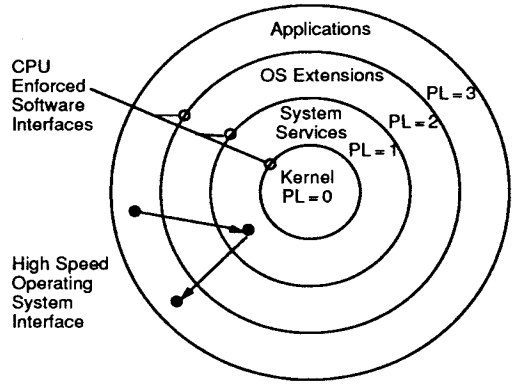


Figure 24. Hierarchical Privilege Levels

Task Privilege

The task always executes at one of the four privilege levels. A 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 when the task is initiated via a task switch operation. 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 executed at 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 privilege level (CPL) at which a task may access the descriptor. 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 numberically 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 of privilege level violation will cause exception 13. A not present fault causes exception 12.

Control Transfer

Four types of control transfer can occur when a selector is loaded into CS by a control transfer operation (see Table 10). 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 descriptor 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 to 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 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, a jump or an interrupt which references either a task gate or task state segment at the same or less privileged level.

Table 9. Descriptor Types Used for Control Transfer

Control Transfer Types	Descriptor Operation Types	Descriptor Referenced	Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL Interrupt Instruction, Exception, External Interrupt	Call Gate Trap or Interrupt Gate	GDT/LDT IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
Task Switch	CALL, JMP CALL, JMP IRET** Interrupt Instruction, Exception, External Interrupt	Task State Segment Task Gate Task Gate	GDT GDT/LDT IDT

* NT (Nested Task bit of flag word) = 0
** NT (Nested Task bit of flag word) = 1

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 intersegment 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 affect the CPU execution state (e.g., HLT) and code or data segments from improper usage. These mechanisms are grouped under the term "protection" and have 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 10), operand reference checks (Table 11), and privileged instruction checks (Table 12). Any violation of the rules shown will result in an exception. A not-present exception related to the stack segment causes exception 12.

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 > IOPL. The IOPL field of the flag word is not changed if CPL > 0.

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

Table 10. 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 to SS	13

Table 11. 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[1]	12 or 13

Note: Carry out in offset calculations is ignored.

Table 12. Privileged Instruction Checks

Error Description	Exception Number
CPL ≠ 0 when executing the following instructions: LIDT, LLDT, LGDT, LTR, LMSW, CTS, HLT	13
CPL > IOPL when executing the following instructions INS, IN, OUTS, OUT, STI, CLI, LOCK	13

Exceptions

The 80C286 detects several types of exceptions and interrupts in protected mode (see Table 13). Most are restartable after the exceptional condition is removed. Interrupt handlers for most exceptions receive 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.

Table 13. Protected Mode Exceptions

Interrupt Vector	Function	Return Address At Failing Instruction?	Always Restartable?	Error Code on Stack?
8	Double exception detected	Yes	No[2]	Yes
9	Processor extension segment overrun	No	No[2]	No
10	Invalid task state segment	Yes	Yes	Yes
11	Segment not present	Yes	Yes	Yes
12	Stack segment overrun or segment not present	Yes	Yes[1]	Yes
13	General protection	Yes	No[2]	Yes

Notes 1. When a PUSHA or POPA 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).

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

3. All these checks are performed for all instructions and can be split into three categories: Segment Load Checks (Table 10), Operand Reference Checks (Table 11), and Privileged Instruction Checks (Table 12). 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 n 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 25) containing the entire 80C286 execution state while a task gate descriptor contains a TSS selector. The limit field must be > 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 function of the IRET instruction. If NT = 0, the IRET instruction performs the regular current task return; 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 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 14). 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 indicates whether use of the selector or segment will cause an exception.

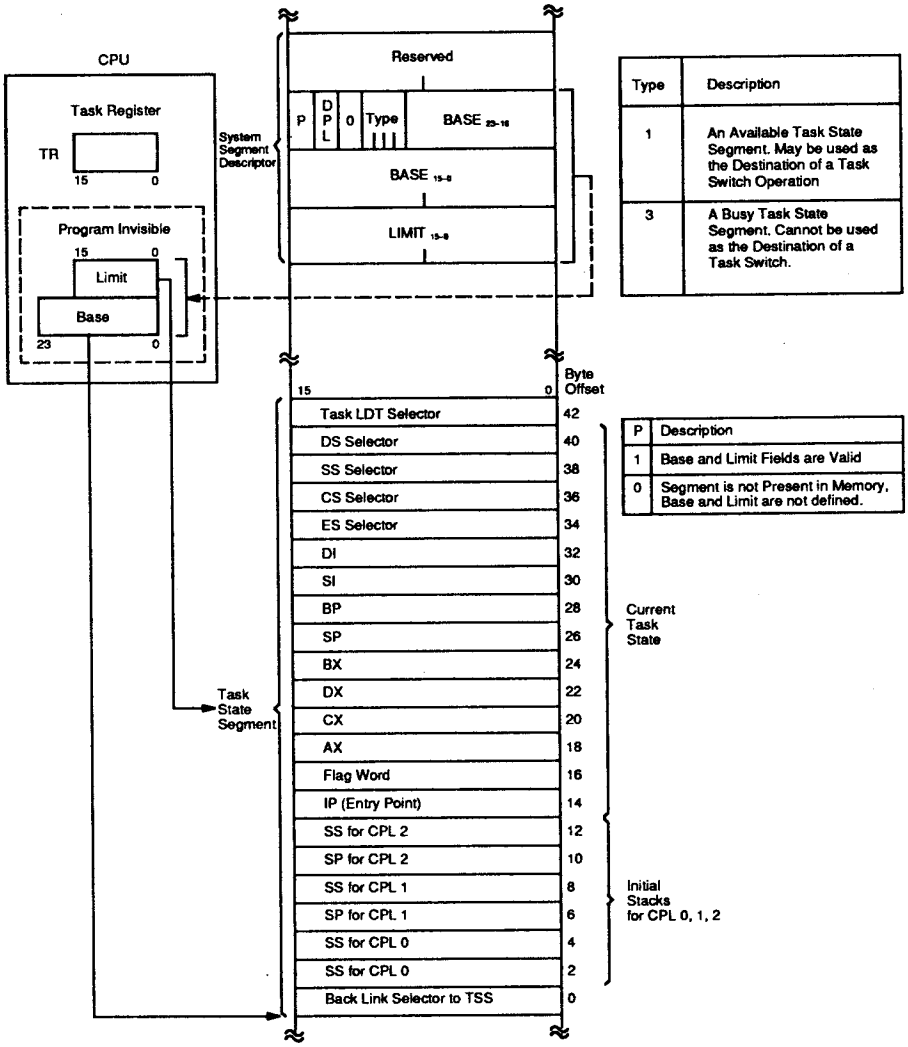


Figure 25. Task State Segment and TSS Registers

Table 14. 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.
VERR	Selector	VERify for Read: sets the zero flag is 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.

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 HIGH.

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 using CS thereafter. The initial CS:IP value of FFOO:FFFO provides 64K bytes of code space for initialization code without changing CS.

Before placing the 80C286 into protected mode, several registers must 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 a task switch operation involves saving the current task state.

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. The 80C286 family includes several devices to generate standard system buses such as the IEEE 796 Standard MULTIBUS®.

Bus Interface Signals and Timing

The 80C286 microsystem local bus interfaces the 80286 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, and peripheral chips performing the duties of clock generator, bus controller, bus arbiter, transceivers, and latches, provide a buffered and decoded system bus interface. A clock generator peripheral chip generates the system clock and synchronizes $\overline{\text{READY}}$ and $\overline{\text{RESET}}$. A bus controller peripheral chip converts bus operation status encoded by the 80C286 into command and bus control signals. These components can provide the timing and electrical power drive levels required for most system bus interfaces including the multibus.

Bus Hold Circuitry

Bus-hold circuitry has been used on 80C286 pins to avoid high current conditions that may be caused by floating inputs to CMOS devices. Figure 25A shows the circuit that will maintain the last valid logic state if no driving source is present (i.e., an unconnected pin or driving source that goes to a high impedance state). Figure 25B shows the circuit that will maintain a high impedance logic state if no driving source is present. In order to overdrive the "bus-hold" circuits, an external driver must be able to sink or source approximately 400 μAmps at valid input voltage levels. Since this bus-hold circuitry is active and not a resistive type element, the power supply current associated with it is negligible.

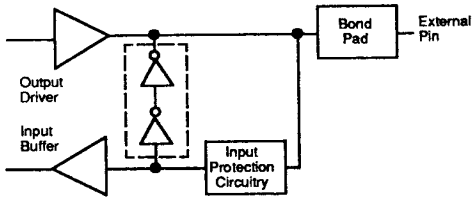


Figure 25A. Bus Hold Circuitry—Pins 36–51, 66, 67

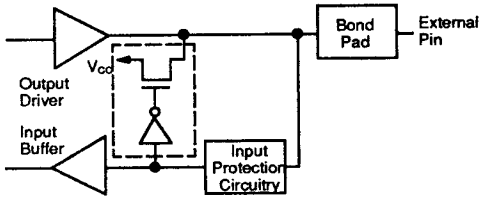


Figure 25B. Bus Hold Circuitry—Pins 4–6, 53, 54, 68

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.

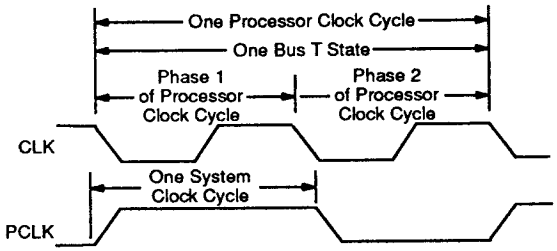
Byte transfers occur on either half of the 16-bit local data bus. Even bytes are accessed over D_{7-0} while odd bytes are transferred over D_{15-8}. Even-addressed words are transferred over D_{15-0} in one bus cycle, while odd-addressed words require two bus operations. The first transfers data on 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 $\overline{BHE}$, control transfers over the lower and upper halves of the data bus. Even address byte transfers are indicated by A_0 LOW and $\overline{BHE}$ HIGH. Odd address byte transfers are indicated by A_0 HIGH and $\overline{BHE}$ LOW. Both A_0 and $\overline{BHE}$ 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 the 8259A 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 82284 clock generator output (PCLK) identifies the next phase of the processor clock. (See Figure 26.)



AF003241

Figure 26. 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 27 shows the four 80C286 local bus states and allowed transitions.

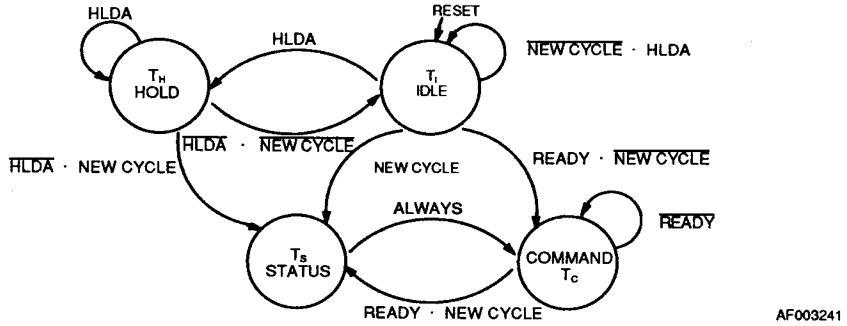


Figure 27. 80C286 Bus States

Bus States

The idle (T_I) state indicates that no data transfers are in progress or requested. The first active state, T_S, is signalled by either status line $\overline{S1}$ or $\overline{S0}$ going LOW also 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 bus controller peripheral chip decodes the status signals and generates Multibus compatible read/write command and local transceiver control signals.

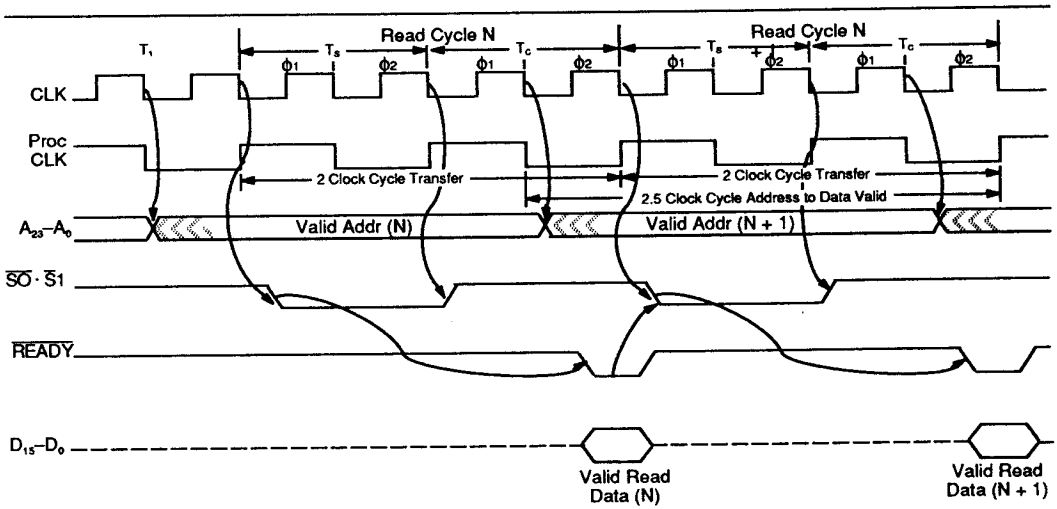
After T_S, the perform command (T_C) state is entered. Memory or I/O devices respond to the bus operation during T_C, either transferring read data to the CPU or accepting write data. T_C states may be repeated as often as necessary to assure sufficient time for the memory or I/O device to respond. The $\overline{READY}$ 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 pins, 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 bus operations to be performed in 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.



Pipelining: valid address (N + 1) available in last phase of bus cycle (N).

WF007840

Figure 28. Basic Bus Cycle

The 80C286 does not maintain the address of the current bus operation during all Tc states. Instead, the address for the next bus operation may be emitted during phase 2 of any Tc. The address remains valid during phase 1 of the first Tc to guarantee hold time, relative to ALE, for the address latch inputs.

Bus Control Signals

The bus controller peripheral chip provides control signals: address latch enable (ALE), Read/Write commands, data transmit/receive (DT/$\overline{R}$), 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 the bus controller peripheral chip outputs Data Enable (DEN) and Data Transmit/Receive (DT/$\overline{R}$). DEN enables the data transceivers while DT/$\overline{R}$ controls transceiver direction. DEN and DT/$\overline{R}$ are timed to prevent bus contention between the bus master, data bus transceivers, and system data bus transceivers.

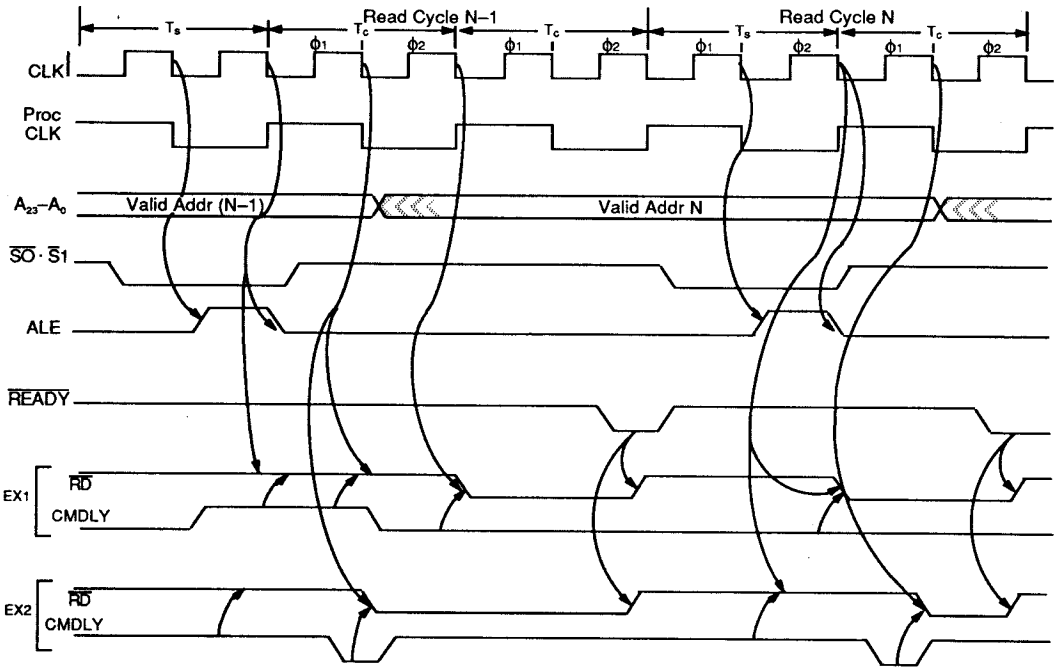
Command Timing Controls

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 8086. External logic can control the duration of any bus operation such that the operation is only as long as necessary. The $\overline{READY}$ input signal can extend any bus operation for as long as necessary.

Command delay allows an increase of address or write data set-up 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 bus controller peripheral chip CMDLY input. After Ts, the bus controller samples CMDLY at each failing edge of CLK. If CMDLY is HIGH, the bus controller peripheral chip will not activate the command signal. When CMDLY is LOW, the bus controller peripheral chip 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/$\overline{R}$.



WF007850

Figure 29. CMDLY Controls and Leading Edge of the Command

Figure 29 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.

Bus Cycle Termination

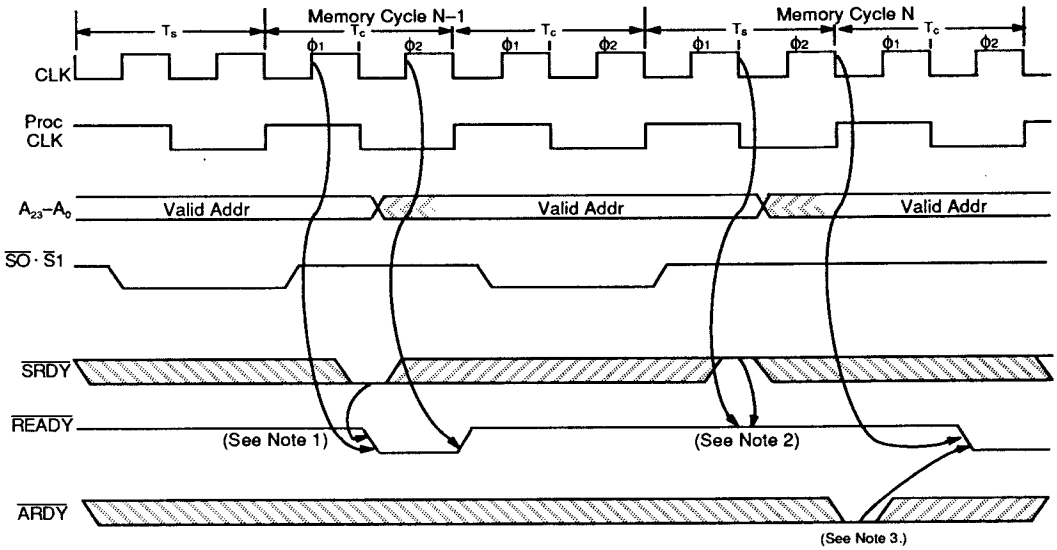
At maximum transfer rates, the 80C286 bus alternates between the status and command states. The bus status signals become inactive after Ts so that they may correctly signal the start of the next bus operation after the completion of the current cycle. No external indication of Tc exists on the 80C286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

$\overline{\text{READY}}$ Operation

The current bus master and bus controller peripheral chip terminate each bus operation simultaneously to achieve maximum bus bandwidth. Both are informed in advance by $\overline{\text{READY}}$ active which identifies the last Tc cycle of the current bus operation. The bus master and bus controller must see the same sense of the $\overline{\text{READY}}$ signal, thereby requiring $\overline{\text{READY}}$ be synchronous to the system clock.

Synchronous Ready

The clock generator peripheral chip provides $\overline{\text{READY}}$ synchronization from both synchronous and asynchronous sources (See Figure 30). The synchronous ready input ($\overline{\text{SRDY}}$) of the clock generator is sampled with the falling edge of CLK at the end of phase 1 of each Tc. The state of $\overline{\text{SRDY}}$ is then broadcast to the bus master and bus controller via the $\overline{\text{READY}}$ output line.



WF007860

Figure 30. Synchronous and Asynchronous Ready

Notes: 1. $\overline{\text{SRDYEN}}$ is active LOW.
2. If $\overline{\text{SRDYEN}}$ is HIGH, the state of $\overline{\text{SRDY}}$ will not effect $\overline{\text{READY}}$.
3. $\overline{\text{ARDYEN}}$ is active LOW.

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 clock generator peripheral chip $\overline{\text{SRDY}}$ set-up and hold time requirements. The clock generator peripheral chip asynchronous ready input ($\overline{\text{ARDY}}$) is designed to accept such signals. The $\overline{\text{ARDY}}$ input is sampled at the beginning of each Tc cycle by 82284 synchronization logic. This provides a system CLK cycle time to resolve its value before broadcasting it to the bus master and bus controller.

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

Each ready input of 82284 has an enable pin ($\overline{\text{SRDYEN}}$ and $\overline{\text{ARDYEN}}$) 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 $\overline{\text{ARDY}}$ or $\overline{\text{SRDY}}$.

Data Bus Control

Figures 31, 32, and 33 show how the DT/$\overline{\text{R}}$, DEN, data bus, and address signals operate for different combinations of read, write, and idle bus operations. DT/$\overline{\text{R}}$ goes active (LOW) for a read operation. DT/$\overline{\text{R}}$ remains HIGH before, during, and between write operations.

The data bus is driven with write data during the second phase of Ts. 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 begins driving the local data bus for write operations. Write data will always remain valid for one system clock past the last Tc 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 three-state OFF during the second phase of the processor cycle after the last Tc. In a write-write sequence the data bus does not enter three-state OFF between Tc and Ts.

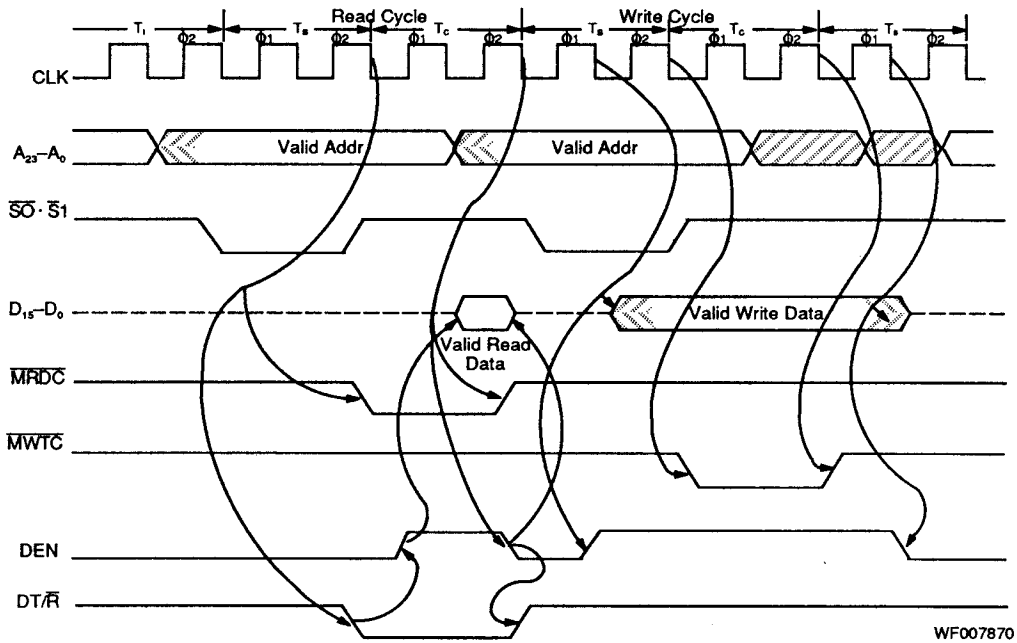


Figure 31. Back-to-Back Read-Write Cycles

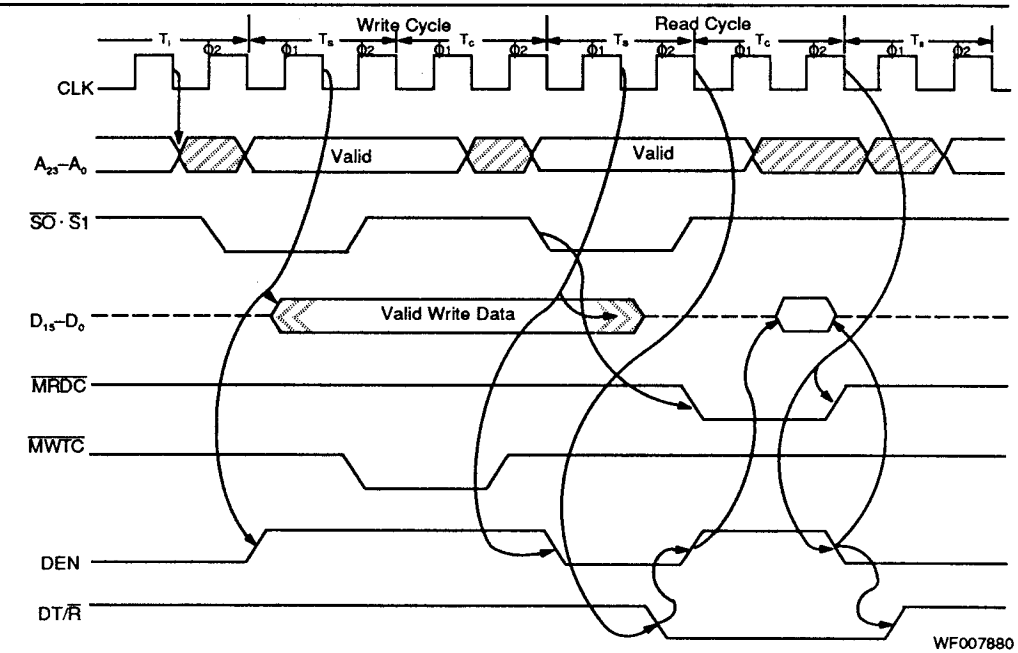


Figure 32. Back-to-Back Write-Read Cycles

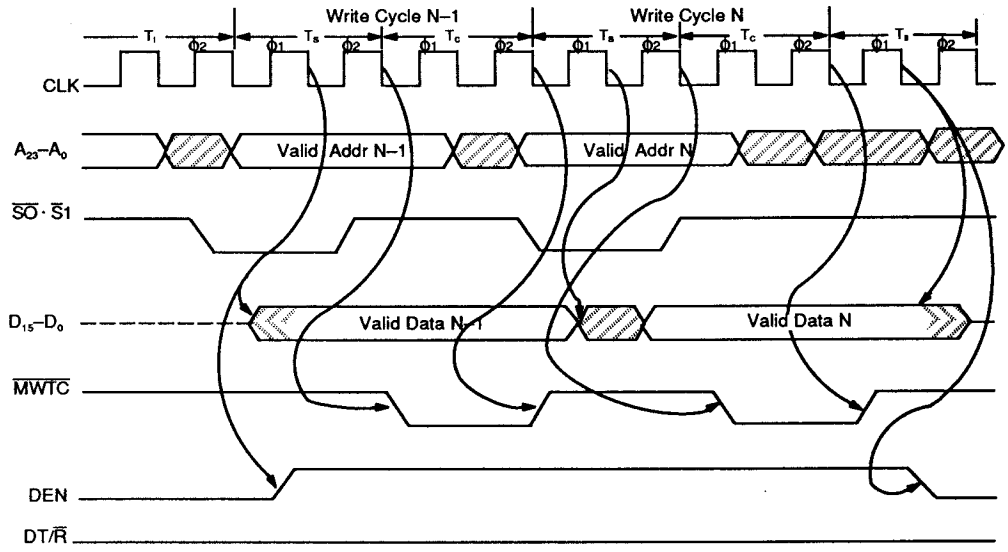


Figure 33. Back-to-Back Write-Write Cycles

1

Bus Usage

The 80C286 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.

HOLD and HLDA

HOLD and HLDA allow another bus master to gain control of the local bus by placing the 80C286 bus into the Th state. The sequence of events required to pass control between the 80C286 and another local bus master are shown in Figure 34.

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_i bus cycle, to guarantee write data hold time, then enters T_h as signaled by HLDA going active.

The 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 5 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.

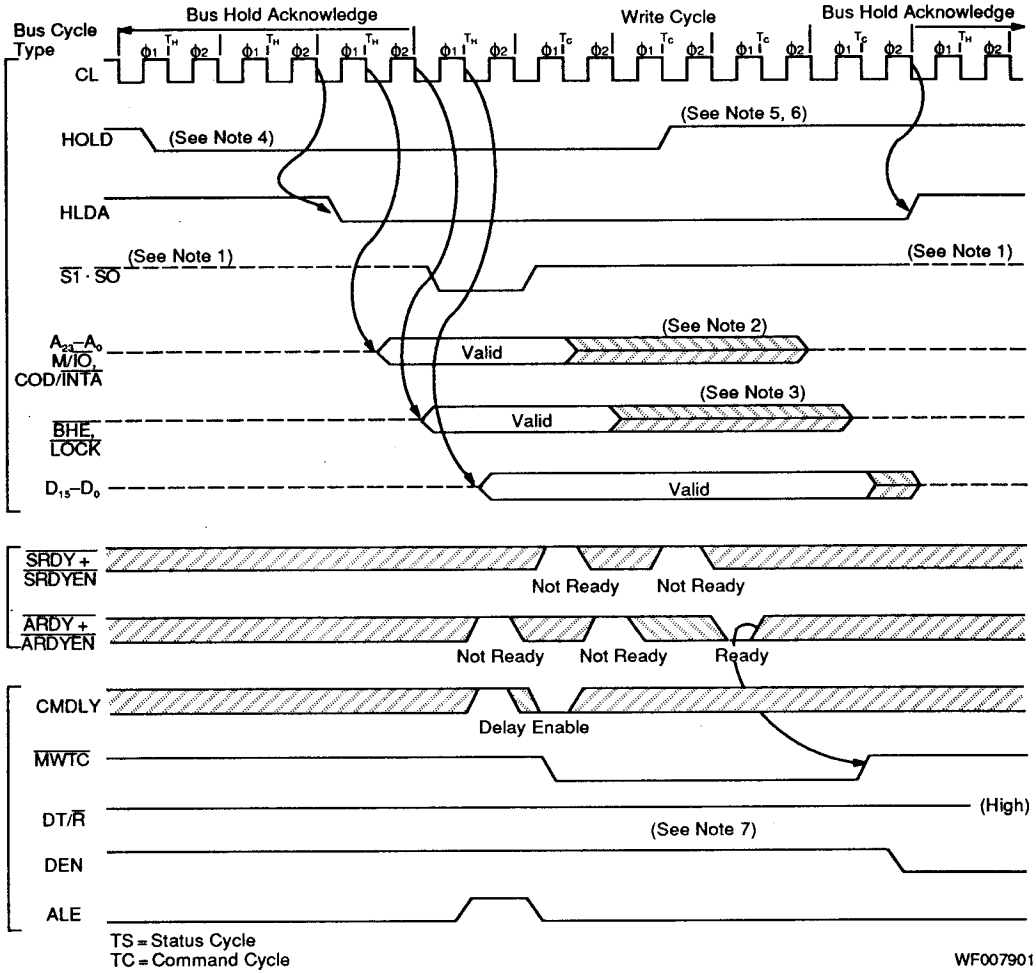


Figure 34. MULTIBUS Write Terminated by Asynchronous Ready with Bus Hold

Notes: 1. Status lines are not driven by 80C286, yet remain high due to pull-up resistors in the peripheral chips during HOLD state.

2. Address, M/IO and COD/INTA may start may start floating during any TC depending on when internal 80C286 bus arbiter decides to release bus to external HOLD. The float starts in Ø2 of TC.
3. BHE and LOCK may start floating after the end of any TC depending on when internal 80286 bus arbiter decides to release bus to external HOLD.
4. The minimum HOLD ↓ to HLDA ↓ time is shown. Maximum is one T_H longer.
5. The earliest HOLD ↑ time is shown which will always allow a subsequent memory cycle if pending.
6. The minimum HOLD ↑ to HLDA ↑ time is shown. Maximum is a function of the instruction, type of bus cycle and other machine status (i.e., Interrupts, Waits, Lock, etc.)
7. 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.

Processor Extension Transfers

The processor extension interface uses I/O port addresses 00F8(H), and 00FA(H), and 00FC(H) which are part of the I/O port address range and is a reserved area. An ESC instruction with EM = 0 and TS = 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, one word transfer with I/O port address 00FA(H), and one or two bus operations with memory are performed. Three bus operations are required for each word operand aligned on an odd byte address.

Interrupt Acknowledge Sequence

Figure 35 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 Programmable Interrupt Controller (PIC) to determine which if any of its slaves should return the interrupt vector. An eight bit vector is read by 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 bus controller peripheral is used to enable the cascade address drivers, during INTA bus operations (see Figure 35), onto the local address bus for distribution to slave interrupt controllers via the system address bus. The 80C286 emits the $\overline{\text{LOCK}}$ 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 interrupt controller peripheral. The second INTA bus operation must always have at least one extra T_c state added via logic controlling $\overline{\text{READY}}$. A_{23}–A_0 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.

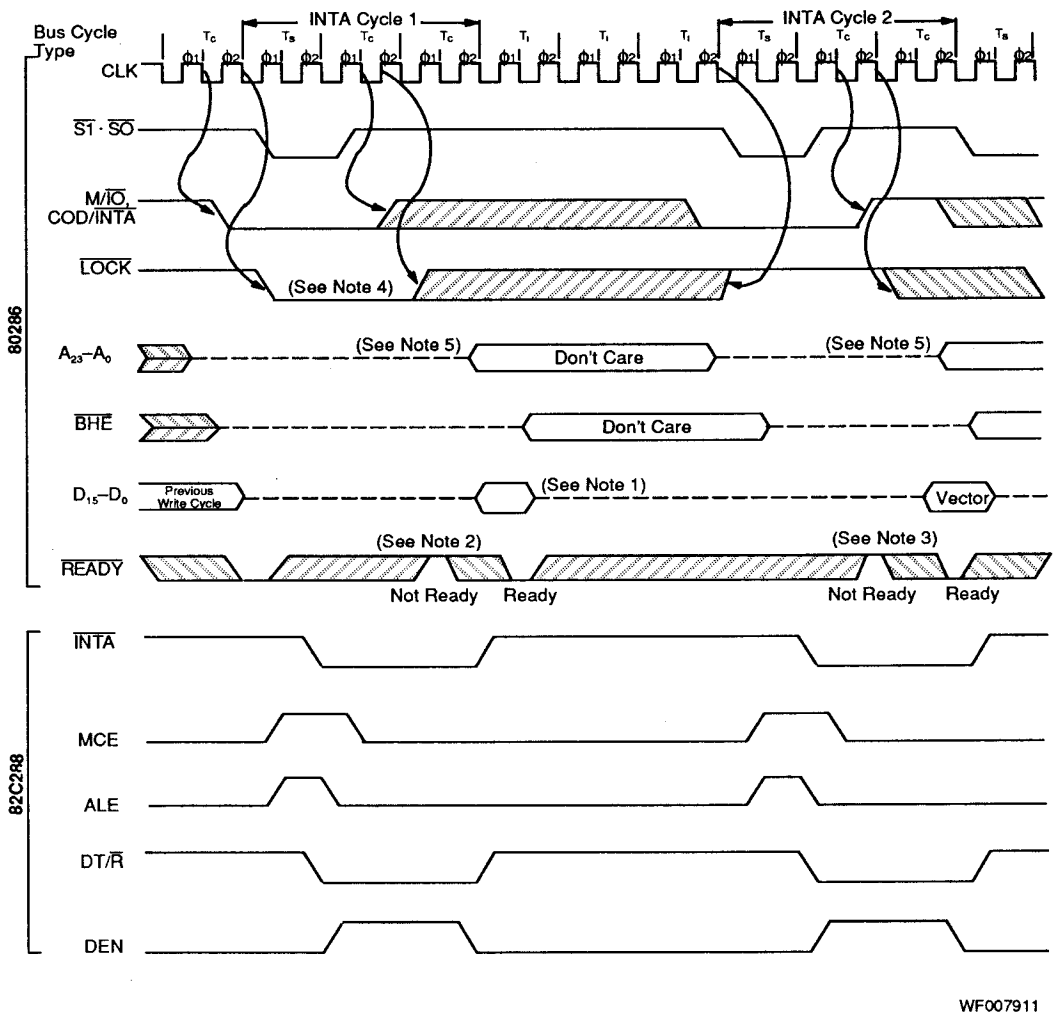


Figure 35. Interrupt Acknowledge Sequence

Notes: 1. Data is ignored.

2. First INTA cycle should have at least one wait state inserted to meet the interrupt controller peripheral minimum INTA pulse width.
3. Second INTA cycle must have at least one wait state inserted since the CPU will not drive A_{23}–A_0, $\overline{BHE}$, and $\overline{LOCK}$ until after the first T_C state.
   The CPU imposed one/clock delay prevents bus contention between cascade address buffer being disabled by MCE ↓ and address outputs.
   Without the wait state, the 80C286 address will not be valid for a memory cycle started immediately after the second INTA cycle. The interrupt controller peripheral also requires one wait state for minimum INTA pulse width.
4. $\overline{LOCK}$ is active for the first INTA cycle to prevent the bus arbiter peripheral from releasing the bus between ITNA cycles in a multi-master system.
5. A_{23}–A_0 exits three-state OFF during Ø2 of the second T_C in the INTA cycle.

Local Bus Usage Priorities

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

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

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

Local bus request via HOLD input.

Processor extension data operand transfer via PEREQ input.

Data transfer performed by EU as part of an instruction.

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

Halt or Shutdown Cycles

The 80C286 externally indicates halt or shutdown conditions as a bus operation. These conditions occur due to a HLT instruction or multiple protection exceptions while attempting to execute one instruction. A halt or shutdown bus operation is signalled when $\overline{S1}$, $\overline{S0}$ and COD/$\overline{INTA}$ are LOW and M/$\overline{IO}$ is HIGH. A_1 HIGH indicates halt, and A_1 LOW indicates shutdown. The bus controller does not issue ALE, nor is $\overline{READY}$ required to terminate a halt or shutdown bus operation.

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

System Configurations

The versatile bus structure of the 80C286 microsystem, with a full complement of support chips, allows flexible configuration of a wide range of systems. The basic configuration, shown in Figure 36, is similar to an iAPX 86 maximum mode system. It includes the CPU plus an interrupt controller, clock generator, and the Bus Controller. The iAPX 86 latches (29843 and 29845) and transceivers (29833 and 29863) may be used in an 80C286 microsystem.

As indicated by the dashed lines in Figure 36, the ability to add processor extensions is an integral feature of 80C286 microsystems. The processor extension interface allows external hardware to perform special functions and transfer data concurrent with CPU execution of other instructions. Full system integrity is maintained because the 80C286 supervises all data transfers and instruction execution for the processor extension.

The 80C286 with the 80C286 numeric processor extension (NPX) uses this interface. The iAPX C286/C287 has all the instructions and data types of an iAPX 86/87 or iAPX 88/87. The 80C286 NPX can perform numeric calculations concurrently with CPU program execution. Numerics code and data have the same integrity as all other information protected by the 80C286 protection mechanism.

The 80C286 can overlap chip select decoding and address propagation during the data transfer for the previous bus operation. This information is latched into the 29843/45s by ALE during the middle of a T_S cycle. The latched chip select and address information remains stable during the bus operation while the next cycle's address is being decoded and propagated into the system. Decode logic can be implemented with a high-speed bipolar PROM.

The optional decode logic shown in Figure 36 takes advantage of the overlap between address and data of the 80C286 bus cycle to generate advanced memory and I/O-select signals. This minimizes system performance degradation caused by address propagation and decode delays. In addition to selecting memory and I/O, the advanced selects may be used with configurations supporting local and system buses to enable the appropriate bus interface for each bus cycle. The COD/INTA and M/IO signals are applied to the decode logic to distinguish between interrupt, I/O, code and data bus cycles.

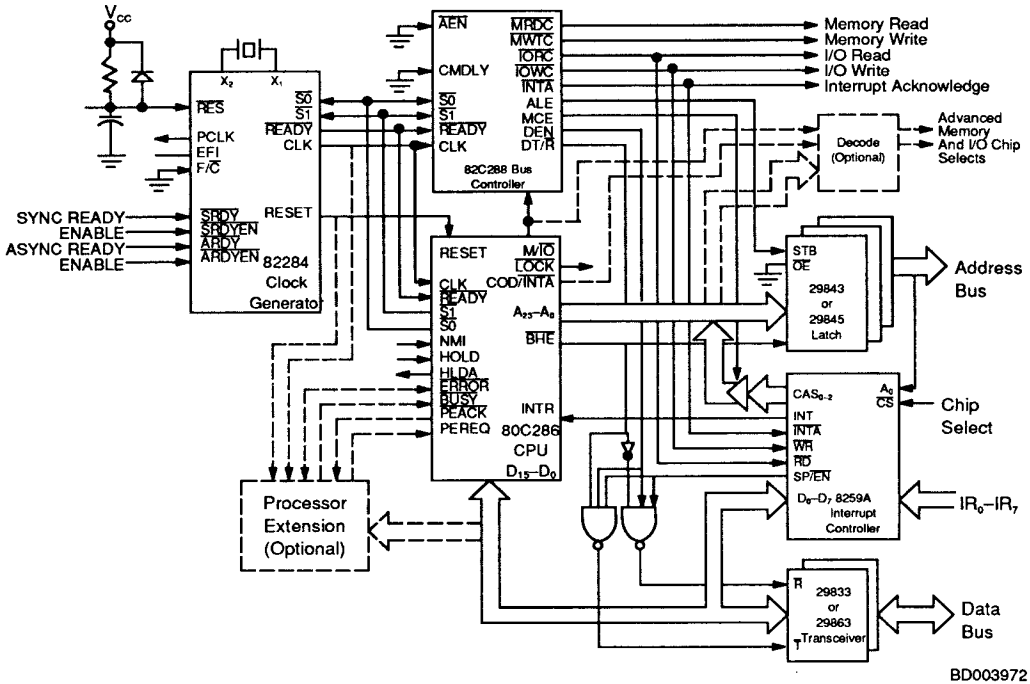


Figure 36. Basic 80C286 System Configuration

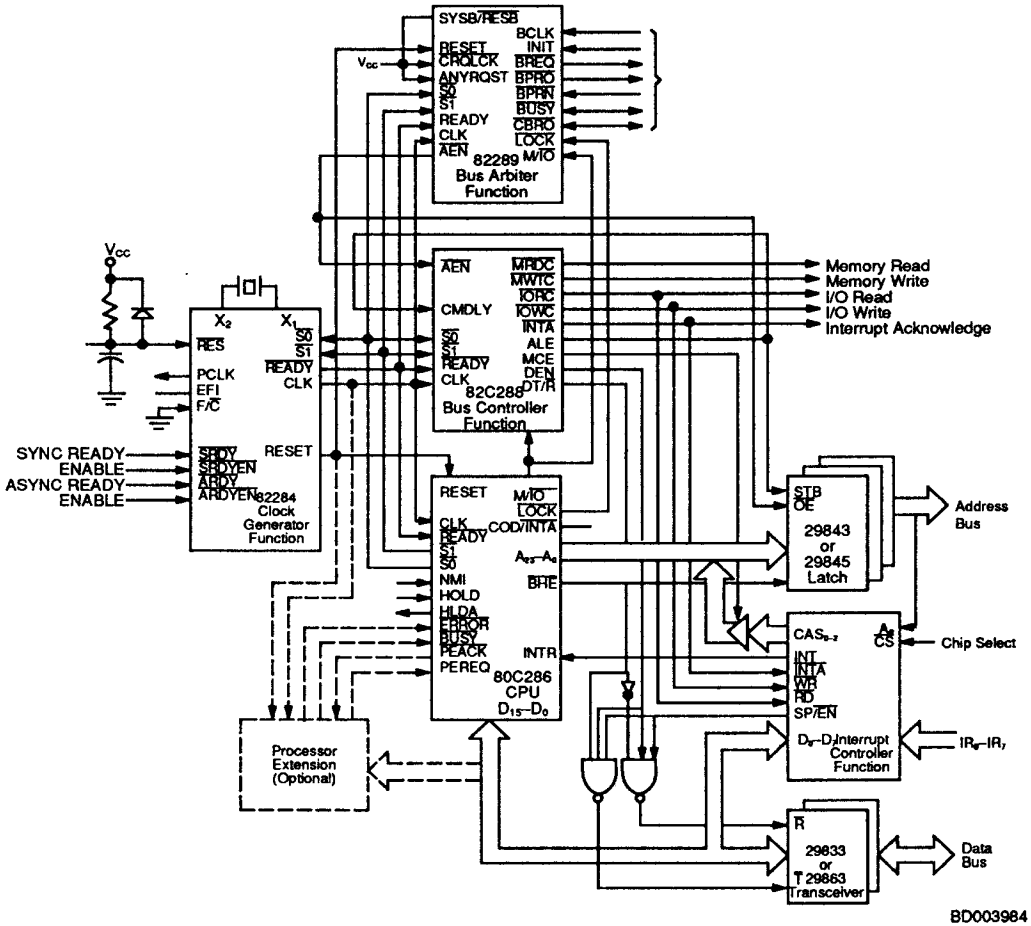
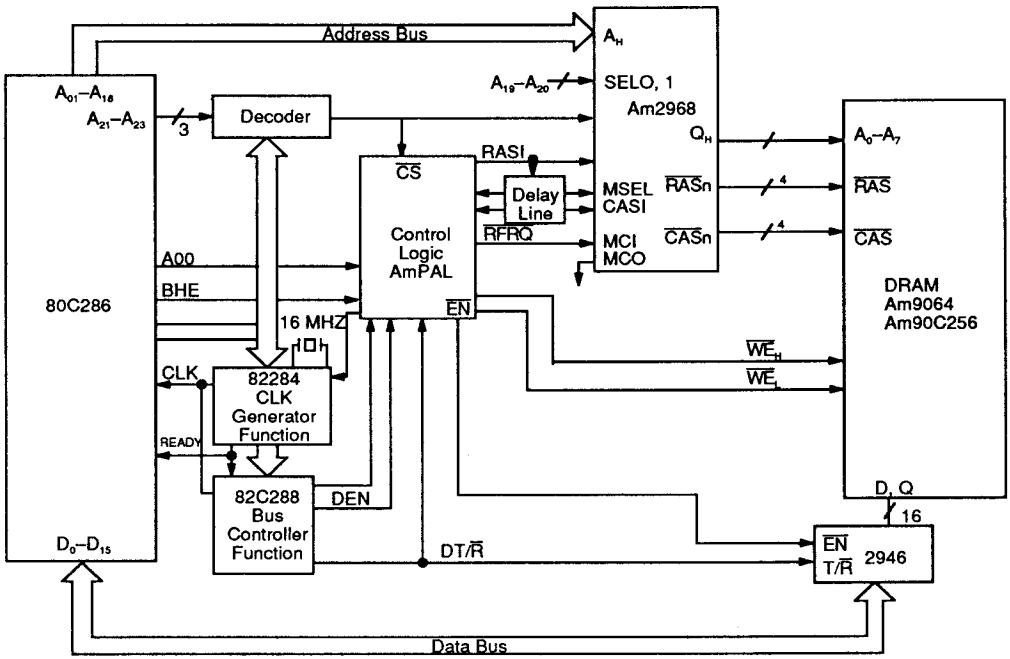


Figure 37. Multibus System Bus Interface

By adding the bus arbiter chip, the 80C286 provides a Multibus system bus interface as shown in Figure 37. The ALE output of the 82C288 for the Multibus bus is connected to its CMDLY input to delay the start of commands one system CLK as required to meet Multibus address and write data set-up times. This arrangement will add at least one extra TC state to each bus operation which uses the Multibus.

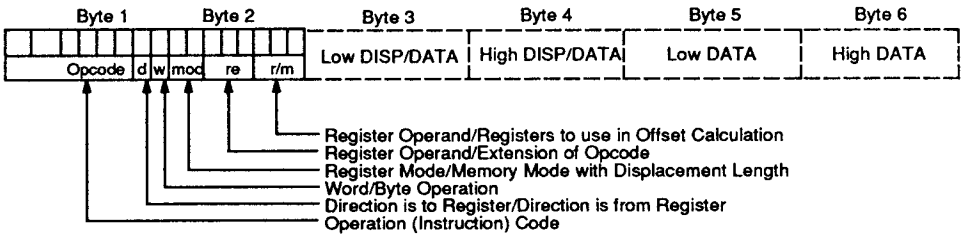
A second bus controller and additional latches and transceivers could be added to the local bus of Figure 37. This configuration allows the 80C286 to support an on-board bus for local memory and peripherals and the Multibus for system bus interfacing.



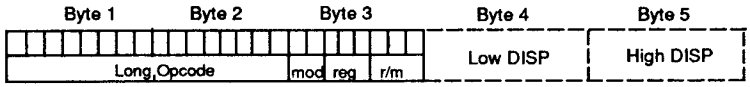
BD005152

Figure 38. 80C286 Interface with the Am2968 Dynamic Memory Controller

Figure 38 shows the interface of the 80286 with the Am2968 Dynamic Memory Controller. The interface is a timing controller which consists of some control logic and a delay line. The timing controller runs asynchronously to the CPU. It arbitrates between memory requests and refresh requests by generating the proper signals to the dynamic memory controller and memory. The design described is a simple, cost-effective solution to interfacing the 80C286 with the Am2968. A further description about DRAM selection based on processor speed may be found in the Am2968 Application Note.



A. Short Opcode Format Example



B. Long Opcode Format Example

Figure 39. 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. A 12.5 processor clock has a clock period of 80 nanoseconds and requires an 80C286 system clock (CLK input) of 25 MHz.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, 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 positive signed value

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 to form the operand

if s = 0 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 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 reference at offset FFFF(H) is attempted.

3. This instruction may be executed in real address mode to initialize the CPU for protected mode.
4. The IOPL and NT fields will remain 0.
5. Processor extension segment overrun interrupt (9) will occur if the operand exceeds the segment limit.

Either Mode

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

Protected Virtual Address Mode Only

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

ABSOLUTE MAXIMUM RATINGS

Supply Voltage ..... +8.0 V
Input, Output or I/O
Voltage Applied GND – 1.V to VDD + 1.0 V
Power Dissipation/Speed
25 MHz ..... 2.1 W
20 MHz ..... 1.7 W
16 MHz ..... 1.4 W
12 MHz ..... 1.2 W
Storage Temperature Range .... –65°C to +150°C
Junction Temperature ..... +175°C
Lead Temperature
(Soldering, Ten Seconds) ..... +275°C

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Operating Voltage Range ..... +4.5 V to +5.5 V
80C286–20 and –25 Only ..... +4.75 V to +5.25 V
Operating Temperature Range .. 0 to +70°C Ambient
(also meets 0 to 100°C Case Temperature for laptop requirements)

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating range

V_{CC} = +5 V ± 10%, for 80C286–12 and 80C286–16, V_{CC} = +5 V ±5% for 80C286–20 and 80C286–25, T_A = 0°C to +70°C

Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Unit
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	I_{OL} = 2.0 mA	–	0.4	V
V_{OH}	Output HIGH Voltage	I_{OH} = –2.0 mA	3.0	–	V
		I_{OH} = –100 µA	V_{CC}–0.4		V
I_I	Input Leakage Current	V_{IN} = GND or V_{CC} Pins 29, 31, 57, 59, 61, 63–64	–10	10	µA
I_{SH}	Input Sustaining Current on $\overline{BUSY}$ and $\overline{ERROR}$ Pins	V_{IN} = GND (see Note 5)	–30	–500	µA
I_{BHL}	Input Sustaining Current HIGH	V_{IN} = 1.0 V (see Note 1)	38	200	µA
I_{BHH}	Input Sustaining Current HIGH	V_{IN} = 3.0 V (see Note 2)	–50	–400	µA
I_O	Output Leakage Current	V_O = GND or V_{CC} Pins 1, 7–8, 10–28, 32–34	–10	10	µA
I_{CCOP}	Active Power Supply Current	80C286–12 (see Note 4)	–	220	mA
		80C286–16 (see Note 4)	–	260	mA
		80C286–20 (see Note 4)	–	310	mA
		80C286–25 (see Note 4)	–	360	mA
I_{CCSB}	Standby Power Supply Current	(see Note 3)	–	5	mA

Notes:

1. I_{BHL} should be measured after lowering V_{IN} to GND and then raising to 1.0 V on the following pins: 36–51, 66, 67.
2. I_{BHH} should be measured after raising V_{IN} to V_{CC} and then lowering to 3.0 V on the following pins: 4–6, 36–51, 66–68.
3. I_{CCSB} tested with the clock stopped in phase two of the processor clock cycle. V_{IN} = V_{CC} or GND, V_{CC} = 5.5 V, outputs unloaded.
4. I_{CCOP} measured at 12.5 MHz for the 80C286–12, 16 MHz for the 80C286–16, and 20 MHz for the 80C286–20. V_{IN} = 2.4 V or 0.4 V, V_{CC} = 5.5 V, outputs unloaded.
5. I_{SH} should be measured after raising V_{IN} to V_{CC} and then lowering to GND on pins 53 and 54.

CAPACITANCE (T_z = +25°C; All Measurements Referenced to Device GND)

Parameter Symbol	Parameter Description	Typ.	Unit	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	

SWITCHING CHARACTERISTICS over operating range

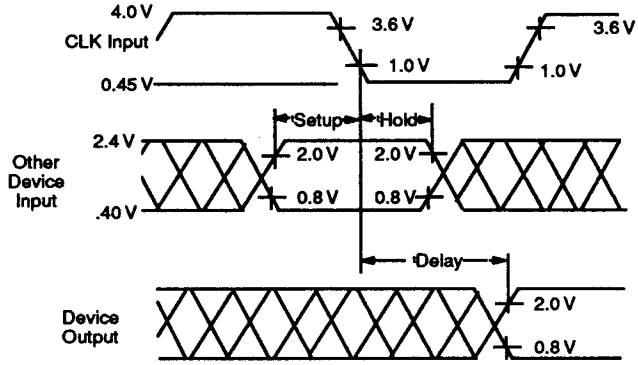
V_{CC} = +5 V ± 10%, T_A = 0°C to +70°C (80C286–12 and 80C286–16)
V_{CC} = +5 V ± 5%, T_A = 0°C to +70°C (80C286–20 and 80C286–25)
AC Timings are referenced to 0.8 V and 2.0 V points of the signals as illustrated in datasheet waveforms, unless otherwise specified.

Parameter Symbol	Parameter Description	Test Conditions	12.5 MHz		16 MHz		Unit
			Min.	Max.	Min.	Max.	
Timing Requirements							
1	System Clock (CLK) Period		40	–	31	–	ns
2	System Clock (CLK) LOW Time	@ 1.0 V	11	–	7	–	ns
3	System Clock (CLK) HIGH Time	@ 3.6 V	13	–	11	–	ns
17	System Clock (CLK) RISE Time	1.0 V to 3.6 V	–	8	–	5	ns
18	System Clock (CLK) FALL Time	3.6 V to 1.0 V	–	8	–	5	ns
4	Asynchronous Inputs SETUP Time	(Note 1)	15	–	5	–	ns
5	Asynchronous Inputs HOLD Time	(Note 1)	15	–	5	–	ns
6	RESET SETUP Time		10	–	10	–	ns
7	RESET HOLD Time		0	–	0	–	ns
8	Read Data SETUP Time		5	–	5	–	ns
9	Read Data HOLD Time		4	–	3	–	ns
10	$\overline{\text{READY}}$ SETUP Time		20	–	12	–	ns
11	$\overline{\text{READY}}$ HOLD Time		20	–	5	–	ns
20	Input RISE/FALL Times	0.8 V to 2.0 V	–	8	–	6	ns
Timing Responses							
12A	Status/$\overline{\text{PEACK}}$ Active Delay	1, (Notes 3, 7)	1	22	1	18	ns
12B	Status/$\overline{\text{PEACK}}$ Inactive Delay	1, (Notes 3, 6)	1	24	1	20	ns
13	Address Valid Delay	1, (Notes 2, 3)	1	32	1	27	ns
14	Write Data Valid Delay	1, (Notes 2, 3)	0	31	0	28	ns
15	Address/Status/Data Float Delay	2, (Note 5)	0	32	0	29	ns
16	HLDA Valid Delay	1, (Notes 3, 8)	0	25	0	25	ns
19	Address Valid to Status SETUP Time	1, (Notes 3, 4)	22	–	16	–	ns

Notes:
1. 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.
2. Delay from 1.0 V on the CLK to 0.8 V or 2.0 V.
3. Output load: C_L = 100 pF.
4. Delay measured from address either reaching 0.8 V or 2.0 V (valid) to status going active reaching 0.8 V or status going inactive reaching 2.0 V.
5. Delay from 1.0 V on the CLK to Float (no current drive) condition.
6. Delay from 1.0 V on the CLK to 0.8 V for min. (HOLD time) and to 2.0 V for max. (inactive delay).
7. Delay from 1.0 V on the CLK to 2.0 V for min. (HOLD time) and to 0.8 V for max. (active delay).
8. Delay from 1.0 V on the CLK to 2.0 V.

Switching Test Conditions

Test Condition	I_L (Constant Current Source	CL
2	[illegible]	[illegible]



AC Setup, Hold and Delay Time Measurement—General WF024251

1

SWITCHING CHARACTERISTICS (continued)

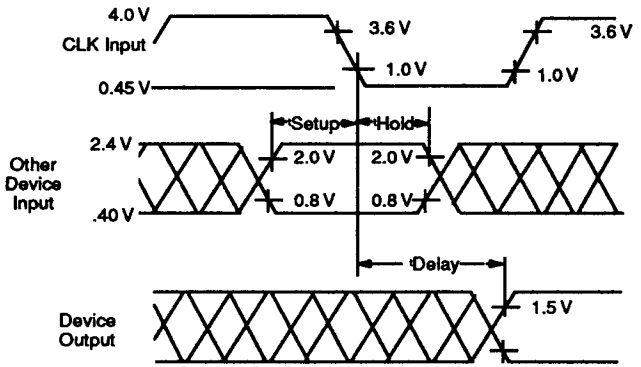
V_{CC} = +5 V ± 10%, T_A = 0°C to +70°C (80C286–12 and 80C286–16)
V_{CC} = +5 V ± 5%, T_A = 0°C to +70°C (80C286–20 and 80C286–25)
AC timings are referenced to 0.8 V and 2.0 V points of the signals as illustrated in datasheet waveforms, unless otherwise specified.

Parameter Symbol	Parameter Description	Test Conditions	20 MHz		25 MHz		Unit
			Min.	Max.	Min.	Max.	
Timing Requirements							
1	System Clock (CLK) Period		25	–	20	–	ns
2	System Clock (CLK) LOW Time	@ 1.0 V	6	–	5	–	ns
3	System Clock (CLK) HIGH Time	@ 3.6 V	9	–	7	–	ns
17	System Clock (CLK) RISE Time	1.0 V to 3.6 V	–	4	–	4	ns
18	System Clock (CLK) FALL Time	3.6 V to 1.0 V	–	4	–	4	ns
4	Asynchronous Inputs SETUP Time	(Note 1)	4	–	4	–	ns
5	Asynchronous Inputs HOLD Time	(Note 1)	4	–	4	–	ns
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	$\overline{\text{READY}}$ SETUP Time		10	–	9	–	ns
11	$\overline{\text{READY}}$ HOLD Time		3	–	3	–	ns
20	Input RISE/FALL Time	0.8 V to 2.0 V	–	6	–	6	ns
Timing Responses							
12A	Status/PEACK Active Delay	1, (Notes 3, 6)	1	15	1	9	ns
12B	Status/PEACK Inactive Delay	1, (Notes 3, 6)	1	16	1	13	ns
13	Address Valid Delay	1, (Notes 2, 3)	1	23	1	18	ns
14	Write Data Valid Delay	1, (Notes 2, 3)	0	27	–	24	ns
15	Address/Status/Data Float Delay	2, (Note 5)	0	25	–	15	ns
16	HLDA Valid Delay	1, (Notes 2, 3)	0	20	–	19	ns
19	Address Valid to Status SETUP Time	1, (Notes 3, 4)	13	–	13	–	ns

Notes:
1. 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.
2. Delay from 1.0 V on the CLK to 0.8 V or 2.0 V.
3. Output load: C_L = 100 pF.
4. Delay measured from address either reaching 0.8 V or 2.0 V (valid) to status going active reaching 0.8 V or status going inactive reaching 2.0 V.
5. Delay from 1.0 V on the CLK to Float (no current drive) condition.
6. Delay from 1.0 V on the CLK to 0.8 V for min. (HOLD time) and to 2.0 V for max. (inactive delay).

Switching Test Conditions

Test Condition	I_L (Constant Current Source	CL
1	2 mA (VOL to Float)	100 pF
2	6 mA (VOL to Float)	100 pF
	8 mA (VOL to Float)	



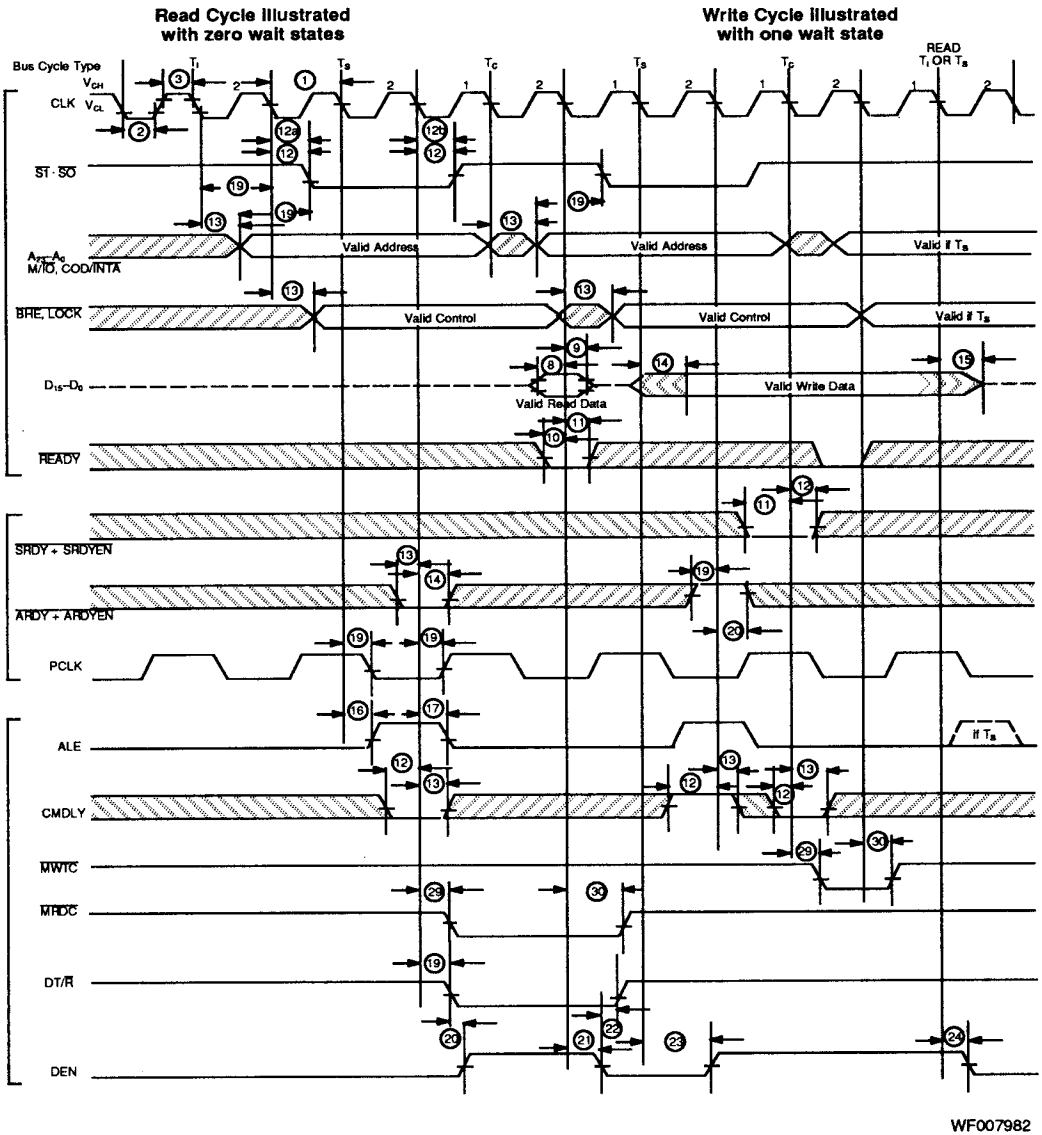
AC Setup, Hold and Delay Time Measurement—For 20/25 MHz only

WF024251

1

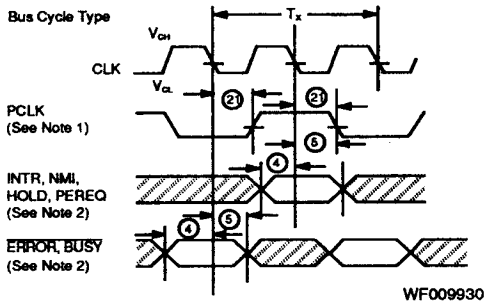
SWITCHING WAVEFORMS

MAJOR CYCLE TIMING



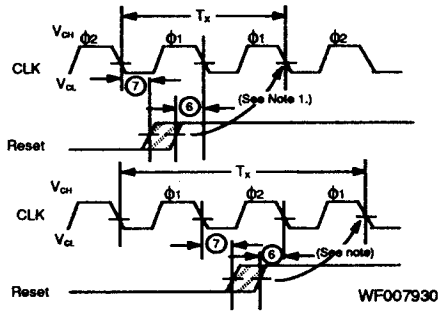
Note: The modified timing is due to the $\overline{\text{CMDLY}}$ signal being active.

80C286 Asynchronous Input Signal Timing



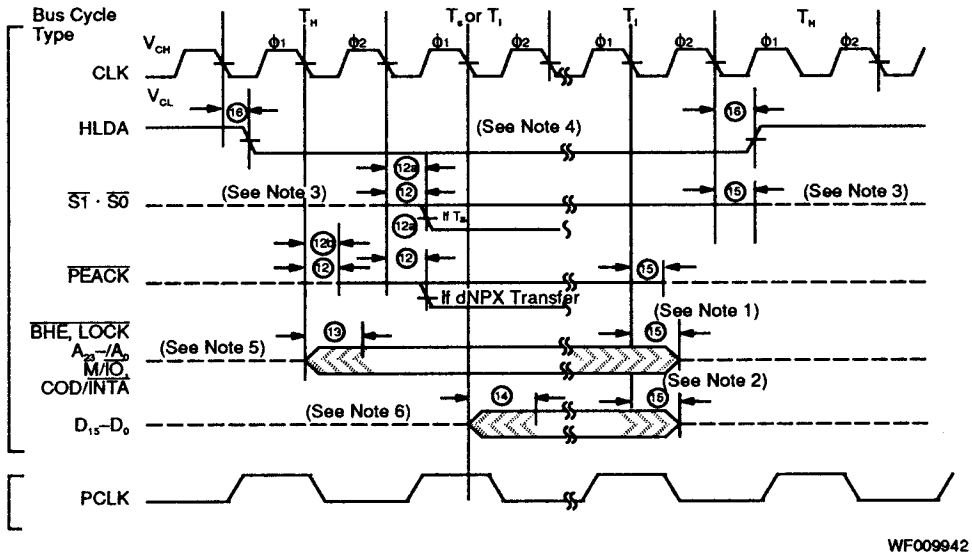
Notes: 1. PCLK indicates which processor cycle phase will occur on the next CLK. PCLK may not indicate the correct phase until the first bus cycle is performed.
2. These inputs are asynchronous. The setup and hold times shown assure recognition for testing purposes.

80C286 Reset Input Timing and Subsequent Processor Cycle Phase



Note When RESET meets the set-up time shown, the next CLK will start or repeat $\phi 1$ of a processor cycle.

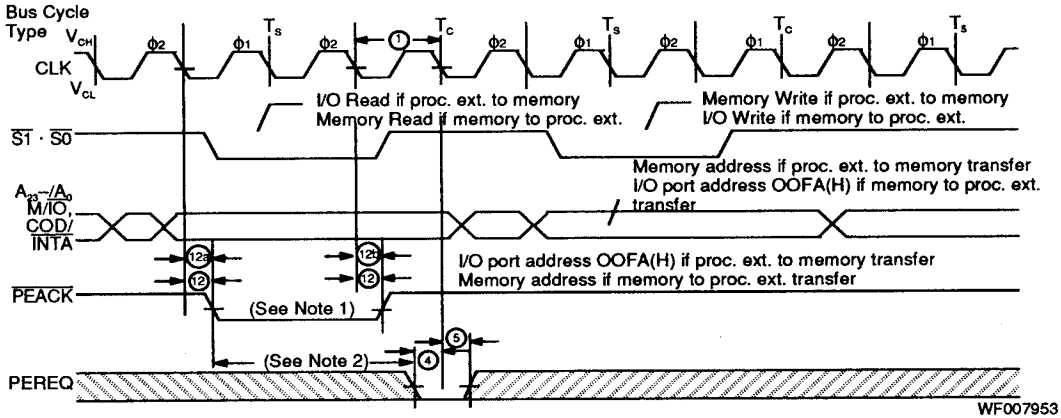
Exiting and Entering Hold



Notes: 1. These signals may not be driven by the 80C286 during the time shown. the worst case in terms of latest float time is shown.
2. The data bus will be driven as shown if the last cycle before T_i in the diagram was a write T_c.
3. The 80C286 floats its status pins during T_H. External 20 kΩ resistors keep these signals high (see Table 15).
4. For HOLD request set-up to HLDA, refer to Figure 34.
5. $\overline{BHE}$ and $\overline{LOCK}$ are driven at this time but will not become valid until T_s.
6. The data bus will remain in three-state OFF if a read cycle is performed.

SWITCHING WAVEFORMS (continued)

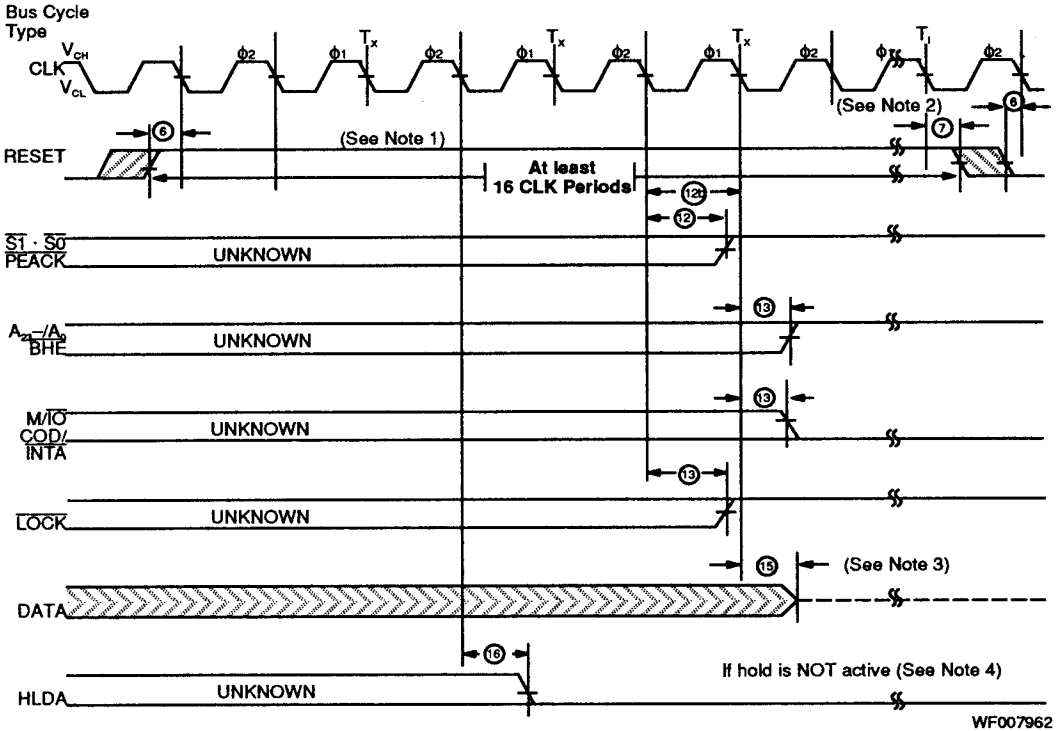
80C286 PEREQ/PEACK Timing Required PEREQ Timing for One Transfer Only



Notes: 1. $\overline{PEACK}$ 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 OOFA(H).

2. To prevent a second processor extension data operand transfer, the worst case maximum time (shown above) is: 3 x 1–12a max–4 min. The actual, configuration dependent, maximum time is: 3 x 1–12a max–4 min + A x 2 x 1. A 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.

Initial 80C286 Pin State During Reset



Notes: 1. Set-up time for RESET ↑ may be violated with the consideration that Ø 1 of the processor clock may begin one system CLK period later.

2. Set-up and hold times for RESET ↓ must be met for proper operation, but RESET ↓ may occur during ϕ1 or ϕ2.

3. The data bus is only guaranteed to be in three-state OFF at the time shown.

4. HOLD is acknowledged during RESET, causing HLDA to go active and the appropriate pins to float. If HOLD remains active while RESET goes inactive, the 80C86 remains in HOLD state and will not perform any bus accesses until HOLD is deactivated.

Function	Format				Clock Count: Real Address Mode	Clock Count: Protected Virtual Address Mode	Comments: Real Address Mode	Comments: Protected Virtual Address Mode
DATA TRANSFER								
MOV = Move:								
Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m			2,3*	2,3*	2	9
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m			2,5*	2,5*	2	9
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1	2,3*	2,3*	2	9
Immediate to register	1 0 1 1 w reg	data	data if w = 1		2	2		
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high		5	5	2	9
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high		3	3	2	9
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m			2,5*	17,19*	2	9,10,11
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m			2,3*	2,3*	2	9
PUSH = Push:								
Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m			2,3*	2,3*	2	9
Register	0 1 0 1 0 reg				3	3	2	9
Segment register	0 0 0 reg 1 1 0				3	3	2	9
Immediate	0 1 1 0 1 0 s 0	data	data if s = 0		3	3	2	9
PUSHA = Push All	0 1 1 0 0 0 0 0				17	17	2	9
POP = Pop								
Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m			5*	5*	2	9
Register	0 1 0 1 1 reg				5	5	2	9
Segment register	0 0 0 reg 1 1 1	(reg ≠ 01)			5	20	2	9,10,11
POPA = Pop All	0 1 1 0 0 0 0 1				19	19	2	9
XCHG = Exchange:								
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m			3,5*	3,5*	2,7	7,9
Register with accumulator	1 0 0 1 0 reg				3	3		
IN = Input from:								
Fixed port	1 1 1 0 1 1 0 w	port			5	5		14
Variable port	1 1 1 0 1 1 0 w				5	5		14
OUT = Output to:								
Fixed port	1 1 1 0 0 1 1 w	port			3	3		14
Variable port	1 1 1 0 1 1 1 w				3	3		14
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1				5	5		9
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m			3*	3*		
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m	(mod p 11)		7*	21*	2	9,10,11
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m	(mod p 11)		7*	21*	2	9,10,11
LAHF = Load AH with flags	1 0 0 1 1 1 1 1				2	2		
SAHF = Store AH into flags	1 0 0 1 1 1 1 0				2	2		
PUSHF = Push flags	1 0 0 1 1 1 0 0				3	3	2	9
POPF = Pop flags	1 0 0 1 1 1 0 1				5	5	2,4	9,15

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Function	Format				Clock Count: Real Address Mode	Clock Count: Protected Virtual Address Mode	Comments: Real Address Mode	Comments: Protected Virtual Address Mode
ARITHMETIC								
ADD=Add:								
Reg/memory with register to either	0 0 0 0 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w = 1		3	3		
Immediate to register	1 0 1 1 w reg	data	data if w = 1		2	2		
ADC = Add with carry:								
Reg/memory with register to either	0 0 0 1 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s:w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w = 1		3	3		
INC =Increment:								
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m			2,7*	2,7*	2	9
Register	0 1 0 0 0 reg				2	2		
SUB =Subtract:								
Reg/memory and register to either	0 0 1 0 1 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s:w = 1	3,7*	3,7*	2	9
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w = 1		3	3		
SBB = Subtract with borrow:								
Reg/memory and register to either	0 0 0 1 1 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s:w = 01	3,7*	3,7*	2	9
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w = 1		3	3		
DEC = Decrement:								
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m			2,7*	2,7*	2	9
Register	0 1 0 0 1 reg				2	2		
CMP = Compare:								
Register/memory with register	0 0 1 1 1 0 1 w	mod 0 0 1 r/m			2,6*	2,6*	2	9
Register with register/memory	0 0 1 1 1 0 0 w	mod reg r/m			2,7*	2,7*	2	9
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s:w = 01	3,6*	3,6*	2	9
Immediate with accumulator	0 0 1 1 1 1 0 w	data	data if w = 1		3	3		
NEG = Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m			2	7*	2	7
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1				3	3		
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1				3	3		
AAS = ASCII adjust for subtract	0 0 1 1 1 1 1 1				3	3		
DAS = Decimal adjust for subtract	0 0 1 0 1 1 1 1				3	3		
MUL = Multiply (unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m						
Register-Byte					13	13		
Register-Word					21	21		
Memory-Byte					16*	16*	2	9
Memory-Word					24*	24*	2	9
IMUL = Integer multiply (signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m						
Register-Byte					13	13		
Register-Word					21	21		
Memory-Byte					16*	16*	2	9
Memory-Word					24*	24*	2	9

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Function	Format	Clock Count: Real Address Mode	Clock Count: Protected Virtual Address Mode	Comments: Real Address Mode	Comments: Protected Virtual Address Mode
ARITHMETIC (Continued)					
IMUL = Integer immediate multiply: (signed)	0 1 1 0 1 0 s 1 | mod reg r/m | data | data if s = 0	21,24*	21,24*	2	9
DIV = Divide (unsigned):	1 1 1 1 0 1 1 w | mod 1 1 0 r/m	2,3*	2,3*	2	9
Register-Byte		14	14		
Register-Word		22	22		
Memory-Byte		17*	17*	2,6	6,9
Memory-Word		25*	25*	2,6	6,9
IDIV = Integer divide (signed)	1 1 1 1 0 1 1 w | mod 1 1 1 r/m				
Register-Byte		17	17		
Register-Word		25	25		
Memory-Byte		20*	20*	2	9
Memory-Word		25*	25*	2	9
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0 | 0 0 0 0 1 0 1 0	16	16		
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1 | 0 0 0 0 1 0 1 0	14	14		
CBW = Convert byte to word	1 0 0 1 1 0 0 0	2	2		
CWD = Convert word to double word	1 0 0 1 1 0 0 1	2	2		
LOGIC					
Shift/Rotate Instructions:					
Register/Memory by 1	1 1 0 1 0 0 0 w | mod TTT r/m	2,7*	2,7*	2	9
Register/Memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	5+n, 8+n*	5+n, 8+n*	2	9
Register Memory by Count	1 1 0 0 0 0 0 w | mod TTT r/m | count	5+n, 8+n*	5+n, 8+n*	2	9

TTT	Instruction
0 0 0	ROL
0 0 1	ROR
0 1 0	RCL
0 1 1	RCR
1 0 0	SHL/SAL
1 0 1	SHR
1 1 1	SAR

AND = And: Reg/memory and register to either	0 0 1 0 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w = 1		3	3		
TEST = And function to flags, no result								
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m			2,6*	2,6*	2	9
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1	3,6*	3,6*	2	9
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w = 1		3	3		
OR = Or:								
Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w = 1		3	3		
XOR = Exclusive or:								
Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w = 1		3	3		
NOT = Invert register/memory	1 1 1 1 0 1 1 w | mod 0 1 0 r/m				2,7*	2,7*	2	9

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Function	Format	Clock Count: Real Address Mode	Clock Count: Protected Virtual Address Mode	Comments: Real Address Mode	Comments: Protected Virtual Address Mode
STRING MANIPULATION:					
MOVS = Move Byte/word	1 0 1 0 0 1 0 w	5	5	2	9
CMPS = Compare byte/word	1 0 1 0 0 1 1 w	8	8	2	9
SCAS = Scan byte/word	1 0 1 0 1 1 1 w	7	7	2	9
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w	5	5	2	9
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w	3	3	2	9
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w	5	5	2	9,14
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w	5	5	2	9,14
Repeated by count in CX					
MOVS = Move string	1 1 1 1 0 0 1 0 | 1 0 1 0 0 1 0 w	5+4n	5+4n	2	9
CMPS = Compare string	1 1 1 1 0 0 1 z | 1 0 1 0 0 1 1 w	5+9n	5+9n	2	9
SCAS = Scan string	1 1 1 1 0 0 1 z | 1 0 1 0 1 1 1 w	5+8n	5+8n	2	9
LODS = Load string	1 1 1 0 0 1 0 | 1 0 1 0 1 1 0 w	5+4n	5+4n	2	9
STOS = Store string	1 1 1 1 0 0 1 0 | 1 0 1 0 1 0 1 w	4+3n	4+3n	2	9
INS = Input string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 0 w	5+4n	5+4n	2	9,14
OUTS = Output string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 1 w	5+4n	5+4n	2	9,14
CONTROL TRANSFER					
CALL = Call:					
Direct within segment	1 1 1 0 1 0 0 0 | disp-low | disp-high	7+m	7+m	2	8
Register memory indirect within segment	1 1 1 1 1 1 1 1 | mod 0 1 0 r/m	7+m,11+m	7+m,11+m	2	8,9
Direct intersegment	1 0 0 1 1 0 1 0 | segment offset | segment selector	13+m	26+m	2	8,11,12
Protected Mode Only (Direct Intersegment):					
Via call gate to same privilege level			41+m		8,11,12
Via call gate to different privilege level, no parameters			82+m		8,11,12
Via call gate to different privilege level, x parameters			86+4x+m		8,11,12
Via TSS			177+m		8,11,12
Via task gate			182+m		8,11,12
Indirect intersegment	1 1 1 1 1 1 1 1 | mod 0 1 1 r/m (mod ≠ 11)	16+m	29+m*	2	8,9,11,12
Protected Mode Only (Indirect Intersegment):					
Via call gate to same privilege level			44+m		8,9,11,12
Via call gate to different privilege level, no parameters			83+m		8,9,11,12
Via call gate to different privilege level, x parameters			90+4x+m		8,9,11,12
Via TSS			180+m		8,9,11,12
Via task gate			185+m		8,9,11,12
JMP = Unconditional jump					
Short/long	1 1 1 0 1 0 1 1 | disp-low	7+m	7+m		8
Direct within segment	1 1 1 0 1 0 0 1 | disp-low | disp-high	7+m	7+m		8
Register/mem indirect within segment	1 1 1 1 1 1 1 1 | mod 1 0 0 r/m	7+m,11+m	7+m,11+m	2	8,9
Direct intersegment	1 1 1 0 1 0 1 0 | segment offset | segment selector	11+m	23+m	2	8,11,12
Protected Mode Only (Indirect Intersegment):					
Via call gate to same privilege level			38+m		8,11,12
Via TSS			175+m		8,11,12
Via task gate			180+m		8,11,12
Indirect intersegment	1 1 1 1 1 1 1 1 | mod 1 0 1 r/m (mod ≠ 11)	15+m	26+m*	2	8,9,11,12

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Function	Format				Clock Count: Real Address Mode	Clock Count: Protected Virtual Address Mode	Comments: Real Address Mode	Comments: Protected Virtual Address Mode
CONTROL TRANSFER (continued):								
Protected Mode Only (Indirect Intersegment)								
Via call gate to same privilege level						41+m*		8,9,11,12
Via TSS						178+m*		8,9,11,12
Via task gate						183+m*		8,9,11,12
RET = Return from CALL:								
Within segment	11000011				11+m	11+m	2	8,9
Within seg adding immed to SP	11000010	data-low	data-high		11+m	11+m	2	8,9
Intersegment	11001011				15+m	25+m	2	8,9,11,12
Intersegment adding immediate to SP	11001010	data-low	data-high		15+m		2	8,9,11,12
Protected Mode Only (RET):								
To different privilege level								
JE/JZ = Jump on equal zero	01110100	disp			7+m or 3	7+m or 3		8
JL/JNGE = Jump on less not greater or equal	01111100	disp			7+m or 3	7+m or 3		8
JLE/JNG = Jump on less or equal not greater	01111110	disp			7+m or 3	7+m or 3		8
JB/JNAE = Jump on below not above or equal	01110010	disp			7+m or 3	7+m or 3		8
JBE/JNA = Jump on below or equal not above	01110110	disp			7+m or 3	7+m or 3		8
JP/JPE = Jump on parity/ parity even	01111010	disp			7+m or 3	7+m or 3		8
JO = Jump on overflow	01110000	disp			7+m or 3	7+m or 3		8
JS = Jump on sign	01111000	disp			7+m or 3	7+m or 3		8
JNE/JNZ = Jump on not equal not zero	01110101	disp			7+m or 3	7+m or 3		8
JNL/JGE = Jump on not less greater or equal	01111101	disp			7+m or 3	7+m or 3		8
JNLE/JG = Jump on not less or equal greater	01111111	disp			7+m or 3	7+m or 3		8
JNB/JAE = Jump on not below above or equal	01110011	disp			7+m or 3	7+m or 3		8
JNBE/JA = Jump on not below or equal above	01110111	disp			7+m or 3	7+m or 3		8
JNP/JPO = Jump on not par/par odd	01111011	disp			7+m or 3	7+m or 3		8
JNO = Jump on not overflow	01110001	disp			7+m or 3	7+m or 3		8
JNS = Jump on not sign	01111001	disp			7+m or 3	7+m or 3		8
LOOP = Loop CX Times	11100010	disp			8+m or 34	8+m or 4		8
LOOPZ/LOOPE = Loop while zero equal	11100001	disp			8+m or 34	8+m or 4		8
LOOPNZ/LOOPNE = Loop while not zero equal	11100000	disp			8+m or 34	8+m or 4		8
JCXZ = Jump on CX zero	11100011	disp			8+m or 34	8+m or 4		8
ENTER = Enter Procedure	11001000	data-low	data-high	L				
L = 0					11	11	2	9
L = 1					15	15	2	9
L > 1					16+4(L−1)	16+4(L−1)	2	9
LEAVE = Leave Procedure	11001001				5	5	2	9

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Function	Format	Clock Count: Real Address Mode	Clock Count: Protected Virtual Address Mode	Comments: Real Address Mode	Comments: Protected Virtual Address Mode
CONTROL TRANSFER (continued):					
INT = Interrupt:					
Type specified	11001101 | type	23+m		2	
Type 3	11001100	23+m		2	
INTO = Interrupt on overflow	11001110	24–m or 3 (3 if no interrupt)	24 – or 3 (3 if no interrupt)	2	
Protected Mode Only:					
Via interrupt or trap gate to same privilege level			40+m		8,11,12
Via interrupt or trap gate to fit different privilege level			78+m		8,11,12
Via Task Gate			167+m		8,11,12
IRET = Interrupt return	11001111	17+m	31+m	2,4	8,9,11, 12,15
Protected Mode Only:					
To different privilege level			55+m		8,9,11, 12,15
To different task (NT = 1)			169+m		8,9,11,12
BOUND = Detect value out of range	01100010 | mod reg r/m	13	13 (Use INT clock count if exception 6	2,6	6,8,9, 11,12
PROCESSOR CONTROL					
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
ESC = Processor Extension Escape	11011TTT | mod LLL r/m	9–20*	9–20*	5	17
	(TTT LL are opcode to processor extension)				
SEG = Segment override prefix	001 reg 110	0	0		

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Function	Format	Clock Count: Real Address Mode	Clock Count: Protected Virtual Address Mode	Comments: Real Address Mode	Comments: Protected Virtual Address Mode
PROTECTION CONTROL:					
LGDT = Load global descriptor table register	00001111 | 00000001 | mod 010 r/m	11*	11*	2,3	9,13
SGDT = Store global descriptor table register	00001111 | 00000001 | mod 000 r/m	11*	11*	2,3	9,13
LIDT = Load interrupt descriptor table register	00001111 | 00000001 | mod 011 r/m	12*	12*	2,3	9,13
SIDT = Store interrupt descriptor table register	00001111 | 00000001 | mod 001 r/m	12*	12*	2,3	9
LLDT = Load local descriptor table register from table memory	00001111 | 00000000 | mod 010 r/m		17,19*	1	9,11,13
SLDT = Store local descriptor table register to register/memory	00001111 | 00000000 | mod 000 r/m		2,3*	1	9
LTR = Load task register from register/memory	00001111 | 00000000 | mod 011 r/m		17,19*	1	9,11,13
STR = Store task register to register memory	00001111 | 00000000 | mod 001 r/m		2,3*	1	9,11,13
LMSW = Load machine status word from register/memory	00001111 | 00000001 | mod 110 r/m	3,6*	3,6*	2,3	9,13
SMSW = Store machine status word	00001111 | 00000001 | mod 100 r/m	2,3*	2,3*	2,3	9
LAR = Load access rights from register/memory	00001111 | 00000010 | mod reg r/m		14,16*	1	9,16
LSL = Load segment limit from register/memory	00001111 | 00000011 | mod reg r/m		14,16*	1	9,16
ARPL = Adjust requested privilege level from register/memory	0 | 01100011 | mod reg r/m		10,11*	2	9
VERR = Verify read access: register/memory	00001111 | 00000000 | mod 100 r/m		14,16*	1	9,16
VERR = Verify write access:	00001111 | 00000000 | mod 101 r/m		14,16*	1	9,16

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

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 mod = 000 then EA = (BX) + (SI) + DISP
if mod = 001 then EA = (BX) + (DI) + DISP
if mod = 010 then EA = (BP) + (SI) + DISP
if mod = 011 then EA = (BP) + (DI) + DISP
if mod = 100 then EA = (SI) + DISP
if mod = 101 then EA = (DI) + DISP
if mod = 110 then EA = (BP) + DISP*
if mod = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)
*except if mod = 00 and r/m = 110 then EA = 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