Chipsmall
深圳市津利帝科技有限公司
Geelead Technology Limited
Chipsmall
采购部

80C286

High Performance Microprocessor with Memory Management and Protection

January 28, 2008

Features

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

Description

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

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

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

Ordering Information

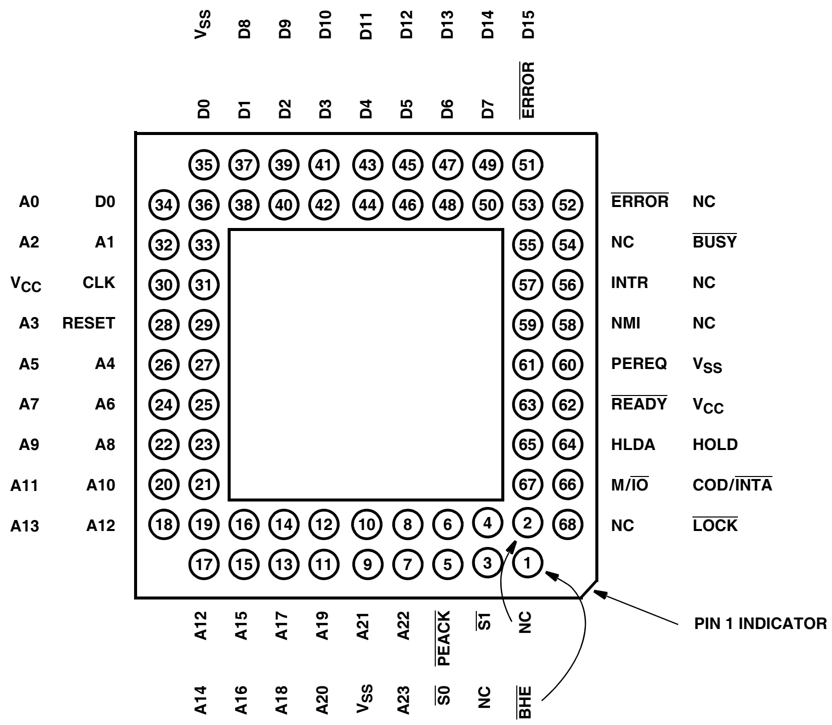
PACKAGE	TEMP. RANGE	10MHz	12.5MHz	16MHz	20MHz	25MHz	PKG. NO.
PGA	$0^{o}C$ to $+70^{o}C$	-	CG80C286-12	CG80C286-16	CG80C286-20	-	G68.B
	$-40^{o}C$ to $+85^{o}C$	IG80C286-10	IG80C286-12	-	-	-	G68.B
	$-55^{o}C$ to $+125^{o}C$	5962-9067801MXC	5962-9067802MXC	-	-	-	G68.B
PLCC	$0^{o}C$ to $+70^{o}C$	-	CS80C286-12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

CAUTION: These devices are sensitive to electrostatic discharge; follow proper IC Handling Procedures.
1-888-INTERSIL or 1-888-468-3774 | Intersil (and design) is a registered trademark of Intersil Americas Inc.

Pinouts

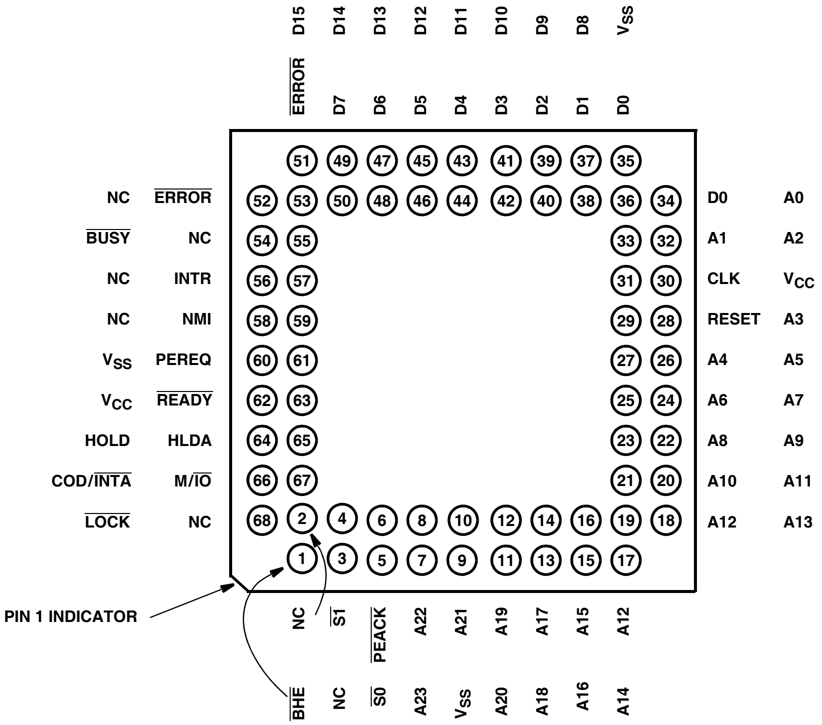
68 LEAD PGA

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



68 LEAD PGA

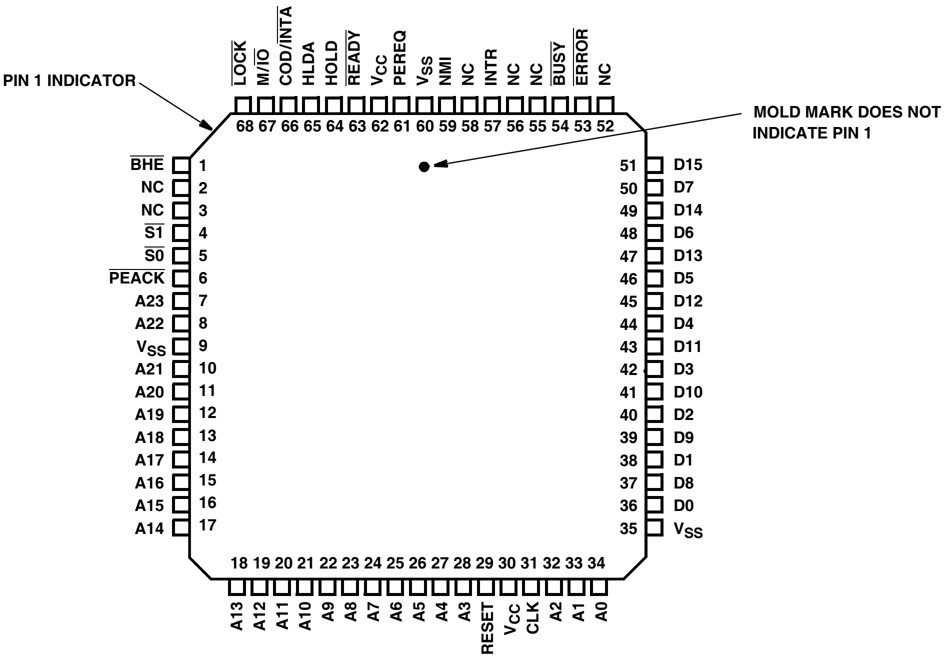
P.C. Board View - As viewed from the component side of the P.C. board.



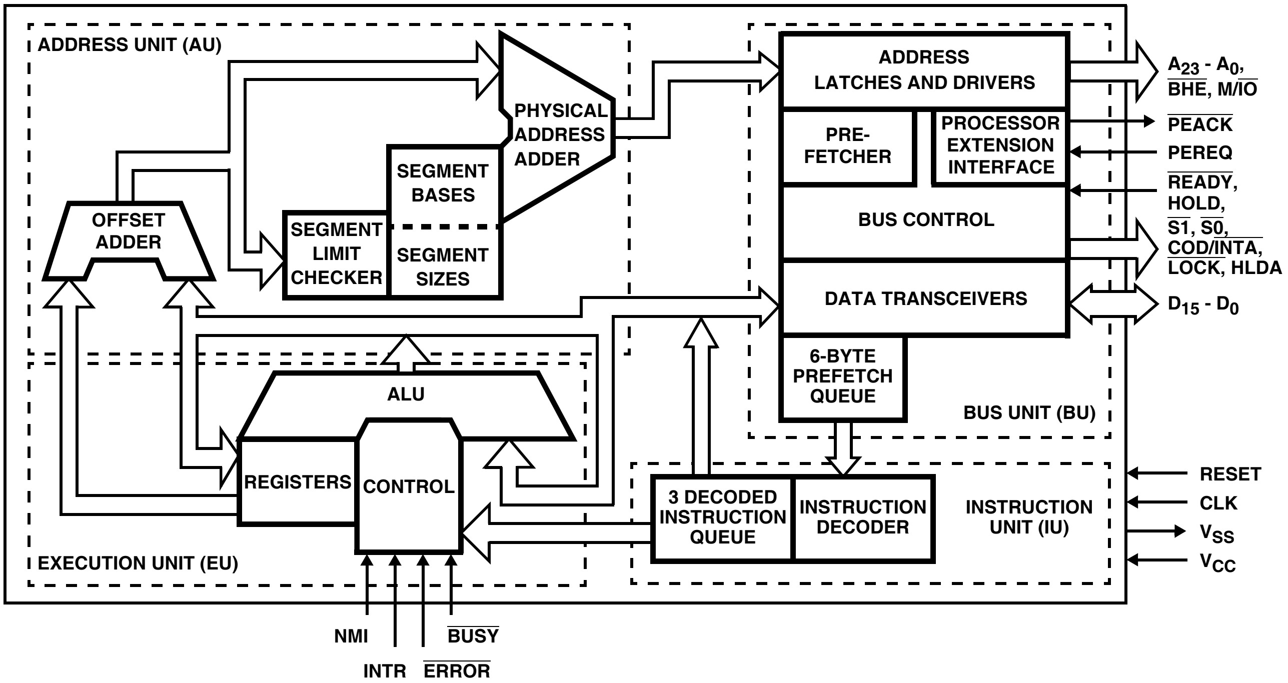
Pinouts (Continued)

68 LEAD PLCC

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



Functional Diagram



Pin Descriptions

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

<table>
<tr><th>SYMBOL</th><th>PIN NUMBER</th><th>TYPE</th><th>DESCRIPTION</th></tr>
<tr><td>CLK</td><td>31</td><td>I</td><td>SYSTEM CLOCK: provides the fundamental timing for the 80C286 system. It is divided by two inside the 80C286 to generate the processor clock. The internal divide-by-two circuitry can be synchronized to an external clock generator by a LOW to HIGH transition on the RESET input.</td></tr>
<tr><td>D_{15} - D_0</td><td>36 - 51</td><td>I/O</td><td>DATA BUS: inputs data during memory, I/O, and interrupt acknowledge read cycles; outputs data during memory and I/O write cycles. The data bus is active HIGH and is held at high impedance to the last valid logic level during bus hold acknowledge.</td></tr>
<tr><td>A_{23} - A_0</td><td>7 - 8
10 - 28
32 - 43</td><td>O</td><td>ADDRESS BUS: outputs physical memory and I/O port addresses. A_{23} - A_{16} are LOW during I/O transfers. A_0 is LOW when data is to be transferred on pins D_7 - D_0 (see table below). The address bus is active High and floats to three-state off during bus hold acknowledge.</td></tr>
<tr><td>$\overline{BHE}$</td><td>1</td><td>O</td><td>BUS HIGH ENABLE: indicates transfer of data on the upper byte of the data bus, D_{15} - D_8. Eight-bit oriented devices assigned to the upper byte of the data bus would normally use BHE to condition chip select functions. BHE is active LOW and floats to three-state OFF during bus hold acknowledge.

$\overline{BHE}$ AND A_0 ENCODINGS

<table>
<tr><th>$\overline{BHE}$ VALUE</th><th>A_0 VALUE</th><th>FUNCTION</th></tr>
<tr><td>0</td><td>0</td><td>Word transfer</td></tr>
<tr><td>0</td><td>1</td><td>Byte transfer on upper half of data bus (D_{15} - D_8)</td></tr>
<tr><td>1</td><td>0</td><td>Byte transfer on lower half of data bus (D_7 - D_0)</td></tr>
<tr><td>1</td><td>1</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>$\overline{S1}$, $\overline{S0}$</td><td>4, 5</td><td>O</td><td>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 T_S state whenever one or both are LOW. $\overline{S1}$ and $\overline{S0}$ are active LOW and are held at a high impedance logic one during bus hold acknowledge.

80C286 BUS CYCLE STATUS DEFINITION

<table>
<tr><th>COD/$\overline{INTA}$</th><th>M/$\overline{IO}$</th><th>$\overline{S1}$</th><th>$\overline{S0}$</th><th>BUS CYCLE INITIATED</th></tr>
<tr><td>0(LOW)</td><td>0</td><td>0</td><td>0</td><td>Interrupt acknowledge</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>Reserved</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>Reserved</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>None; not a status cycle</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>If A_1 = 1 then halt; else shutdown</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>Memory data read</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>Memory data write</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>None; not a status cycle</td></tr>
<tr><td>1(HIGH)</td><td>0</td><td>0</td><td>0</td><td>Reserved</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>I/O read</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>I/O write</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>None; not a status cycle</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>Reserved</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>Memory instruction read</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>Reserved</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>None; not a status cycle</td></tr>
</table>
</td></tr>
</table>

Pin Descriptions

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

SYMBOL	PIN NUMBER	TYPE	DESCRIPTION
M/$\overline{\text{IO}}$	67	O	MEMORY I/O SELECT: distinguishes memory access from I/O access. If HIGH during T_S, a memory cycle or a halt/shutdown cycle is in progress. If LOW, an I/O cycle or an interrupt acknowledge cycle is in progress. M/$\overline{\text{IO}}$ is held at high impedance to the last valid logic state during bus hold acknowledge.
COD/$\overline{\text{INTA}}$	66	O	CODE/INTERRUPT ACKNOWLEDGE: distinguishes instruction fetch cycles from memory data read cycles. Also distinguishes interrupt acknowledge cycles from I/O cycles. COD/$\overline{\text{INTA}}$ is held at high impedance to the last valid logic state during bus hold acknowledge. Its timing is the same as M/$\overline{\text{IO}}$.
$\overline{\text{LOCK}}$	68	O	BUS LOCK: indicates that other system bus masters are not to gain control of the system bus for the current and following bus cycles. The $\overline{\text{LOCK}}$ 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. $\overline{\text{LOCK}}$ is active LOW and is held at a high impedance logic one during bus hold acknowledge.
$\overline{\text{READY}}$	63	I	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 setup and hold times relative to the system clock be met for correct operation. $\overline{\text{READY}}$ is ignored during bus hold acknowledge. (See Note 1)
HOLD HLDA	64 65	I O	BUS HOLD REQUEST AND HOLD ACKNOWLEDGE: control ownership of the 80C286 local bus. The HOLD input allows another local bus master to request control of the local bus. When control is granted, the 80C286 will float its bus drivers and then activate HLDA, thus entering the bus hold acknowledge condition. The local bus will remain granted to the requesting master until HOLD becomes inactive which results in the 80C286 deactivating HLDA and regaining control of the local bus. This terminates the bus hold acknowledge condition. HOLD may be asynchronous to the system clock. These signals are active HIGH. Note that HLDA never floats.
INTR	57	I	INTERRUPT REQUEST: requires the 80C286 to suspend its current program execution and service a pending external request. Interrupt requests are masked whenever the interrupt enable bit in the flag word is cleared. When the 80C286 responds to an interrupt request, it performs two interrupt acknowledge bus cycles to read an 8-bit interrupt vector that identifies the source of the interrupt. To ensure program interruption, INTR must remain active until an interrupt acknowledge bus cycle is initiated. INTR is sampled at the beginning of each processor cycle and must be active HIGH at least two processor cycles before the current instruction ends in order to interrupt before the next instruction. INTR is level sensitive, active HIGH, and may be asynchronous to the system clock.
NMI	59	I	NON-MASKABLE INTERRUPT REQUEST: interrupts the 80C286 with an internally supplied vector value of two. No interrupt acknowledge cycles are performed. The interrupt enable bit in the 80C286 flag word does not affect this input. The NMI input is active HIGH, may be asynchronous to the system clock, and is edge triggered after internal synchronization. For proper recognition, the input must have been previously LOW for at least four system clock cycles and remain HIGH for at least four system clock cycles.
PEREQ $\overline{\text{PEACK}}$	61 6	I O	PROCESSOR EXTENSION OPERAND REQUEST AND ACKNOWLEDGE: extend the memory management and protection capabilities of the 80C286 to processor extensions. The PEREQ input requests the 80C286 to perform a data operand transfer for a processor extension. The $\overline{\text{PEACK}}$ output signals the processor extension when the requested operand is being transferred. PEREQ is active HIGH. $\overline{\text{PEACK}}$ is active LOW and is held at a high impedance logic one during bus hold acknowledge. PEREQ may be asynchronous to the system clock.
$\overline{\text{BUSY}}$ $\overline{\text{ERROR}}$	54 53	I I	PROCESSOR EXTENSION BUSY AND ERROR: indicates 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.

Pin Descriptions

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

<table>
<tr><th>SYMBOL</th><th>PIN NUMBER</th><th>TYPE</th><th>DESCRIPTION</th></tr>
<tr><td>RESET</td><td>29</td><td>I</td><td>SYSTEM RESET: clears the internal logic of the 80C286 and is active HIGH. The 80C286 may be reinitialize at any time with a LOW to HIGH transition on RESET which remains active for more than 16 system clock cycles. During RESET active, the output pins of the 80C286 enter the state shown below.
<b>80C286 PIN STATE DURING RESET</b>
<table>
<tr><th>PIN VALUE</th><th>PIN NAMES</th></tr>
<tr><td>1 (HIGH)</td><td>$\overline{S0}$, $\overline{S1}$, $\overline{PEACK}$, A_{23} - A_0, $\overline{BHE}$, $\overline{LOCK}$</td></tr>
<tr><td>0 (LOW)</td><td>M/$\overline{IO}$, COD/$\overline{INTA}$, HLDA (Note 2)</td></tr>
<tr><td>HIGH IMPEDANCE</td><td>D15 - D0</td></tr>
</table>
Operation of the 80C286 begins after a HIGH to LOW transition on RESET. The HIGH to LOW transition of RESET must be synchronous to the system clock. Approximately 50 system clock cycles are required by the 80C286 for internal initializations before the first bus cycle to fetch code from the power-on execution address is performed. A LOW to HIGH transition of RESET synchronous to the system clock will end a processor cycle at the second HIGH to LOW transition of the system clock. The LOW to HIGH transition of RESET may be asynchronous to the system clock; however, in this case it cannot be predetermined which phase of the processor clock will occur during the next system clock period. Synchronous LOW to HIGH transitions of RESET are required only for systems where the processor clock must be phase synchronous to another clock.</td></tr>
<tr><td>V_{SS}</td><td>9, 35, 60</td><td>I</td><td>SYSTEM GROUND: are the ground pins (all must be connected to system ground).</td></tr>
<tr><td>V_{CC}</td><td>30, 62</td><td>I</td><td>SYSTEM POWER: +5V power supply pins. A 0.1μF capacitor between pins 60 and 62 is recommended.</td></tr>
</table>

NOTES:

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

Functional Description

Introduction

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

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

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

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

Static Operation

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

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

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

80C286 Base Architecture

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

Register Set

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

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

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

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

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

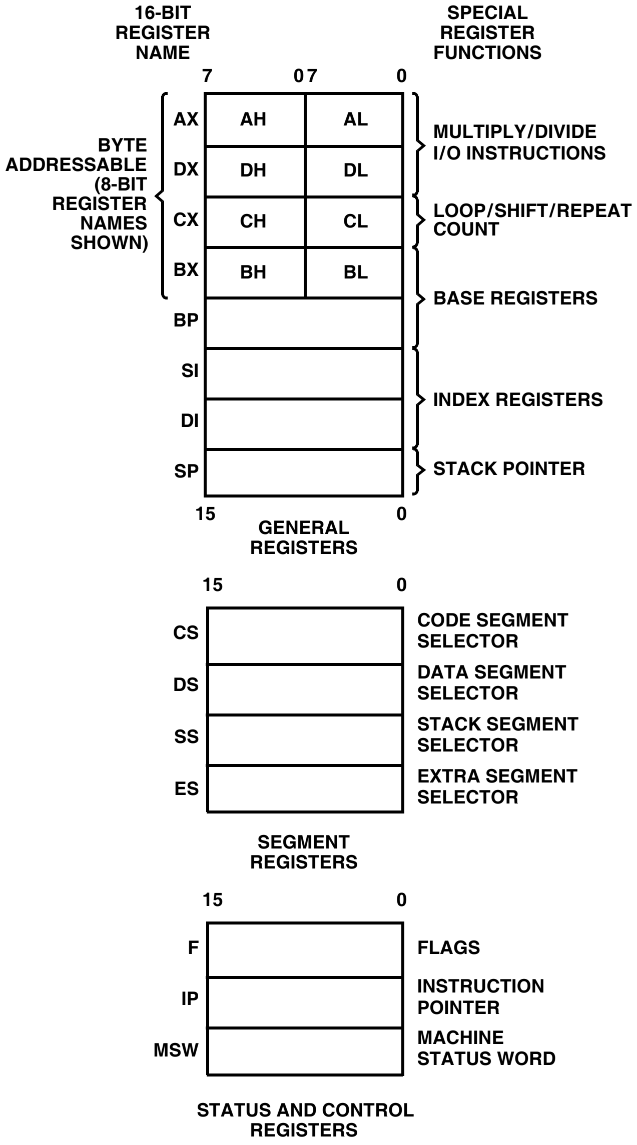


FIGURE 1. REGISTER SET

Flags Word Description

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

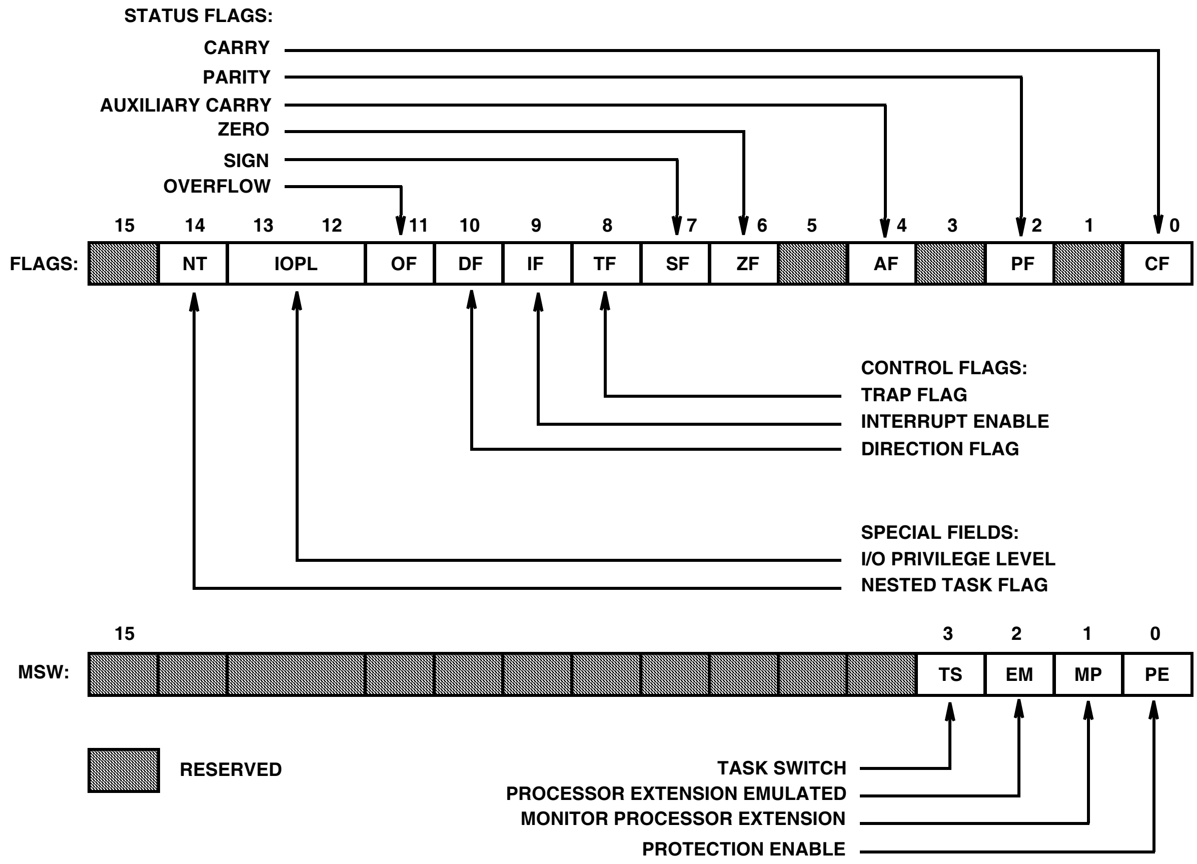


FIGURE 2. STATUS AND CONTROL REGISTER BIT FUNCTIONS

TABLE 1. FLAGS WORD BIT FUNCTIONS

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

Instruction Set

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

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

- Register to Register
- Memory to 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.

TABLE 2A. DATA TRANSFER INSTRUCTIONS

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

TABLE 2B. ARITHMETIC INSTRUCTIONS

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

TABLE 2C. STRING INSTRUCTIONS

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

TABLE 2D. SHIFT/ROTATE LOGICAL INSTRUCTIONS

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

TABLE 2E. HIGH LEVEL INSTRUCTIONS

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

TABLE 2F. PROCESSOR CONTROL INSTRUCTIONS

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

TABLE 2G. PROGRAM TRANSFER INSTRUCTIONS

CONDITIONAL TRANSFERS		UNCONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal	CALL	Call procedure
JAE/JNB	Jump if above or equal/not below	RET	Return from procedure
JB/JNAE	Jump if below/not above nor equal	JMP	Jump
JBE/JNA	Jump if below or equal/not above		
JC	Jump if carry	ITERATION CONTROLS	
JE/JZ	Jump if equal/zero	LOOP	Loop
JG/JNLE	Jump if greater/not less nor equal		
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	INT	Interrupt
JNP/JPO	Jump if not parity/parity odd		
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		

Memory Organization

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

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

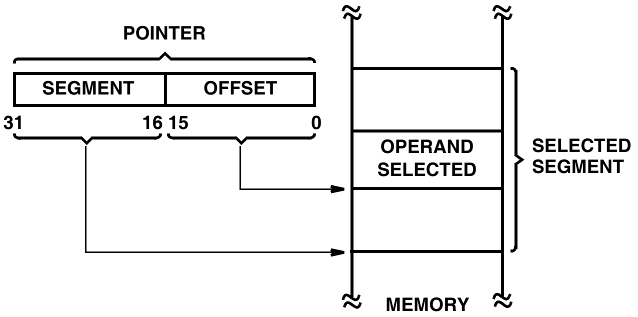


FIGURE 3. TWO COMPONENT ADDRESS

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

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

TABLE 3. SEGMENT REGISTER SELECTION RULES

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

Addressing Modes

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

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

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

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

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

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

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

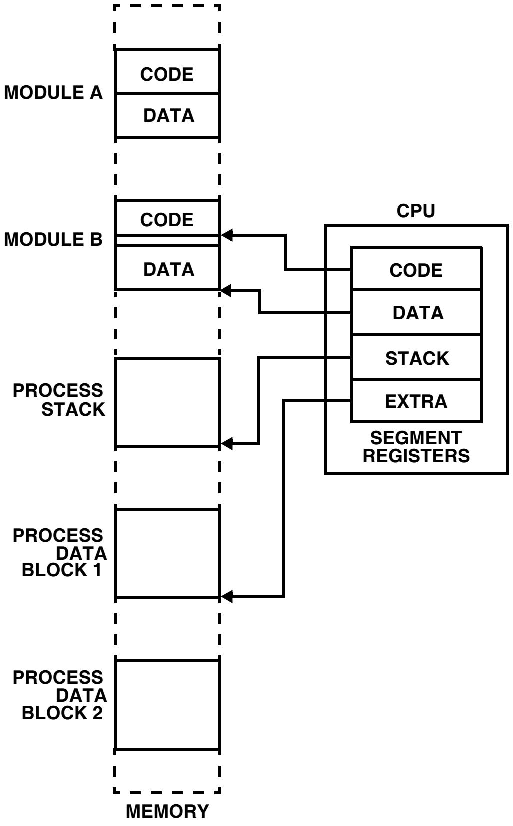


FIGURE 4. SEGMENTED MEMORY HELPS STRUCTURE SOFTWARE

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

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

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

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

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

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

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

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

Data Types

The 80C286 directly supports the following data types:

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

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

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

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

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

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

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

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

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

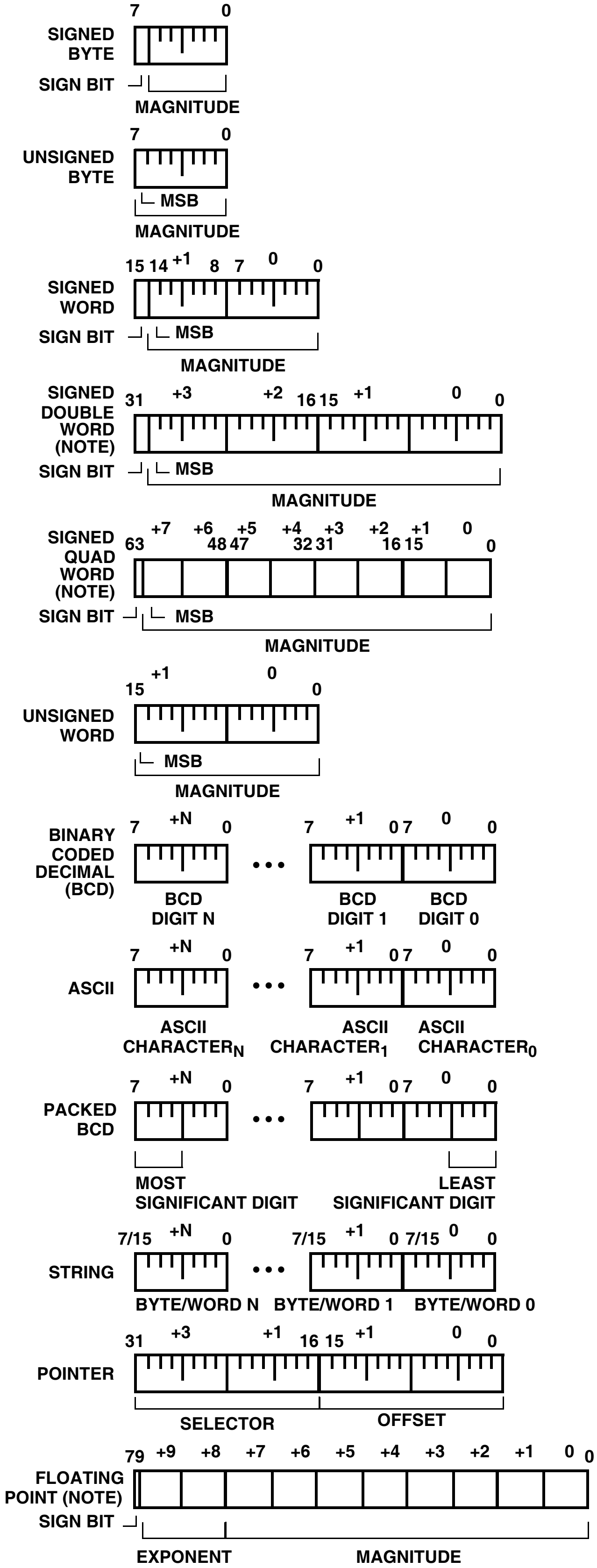


FIGURE 5. 80C286 SUPPORTED DATA TYPES

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

TABLE 4. INTERRUPT VECTOR ASSIGNMENTS

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

I/O Space

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

Interrupts

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

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

Maskable Interrupt (INTR)

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

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

Non-Maskable Interrupt Request (NMI)

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

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

Single Step Interrupt

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

Interrupt Priorities

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

TABLE 5. INTERRUPT PROCESSING ORDER

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

Initialization and Processor Reset

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

TABLE 6. 80C286 INITIAL REGISTER STATE AFTER RESET

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

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

Machine Status Word Description

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

TABLE 7. MSW BIT FUNCTIONS

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

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

Halt

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

TABLE 8. RECOMMENDED MSW ENCODINGS FOR PROCESSOR EXTENSION CONTROL

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

TABLE 9. REAL ADDRESS MODE ADDRESSING INTERRUPTS

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

80C286 Real Address Mode

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

Memory Size

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

Memory Addressing

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

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

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

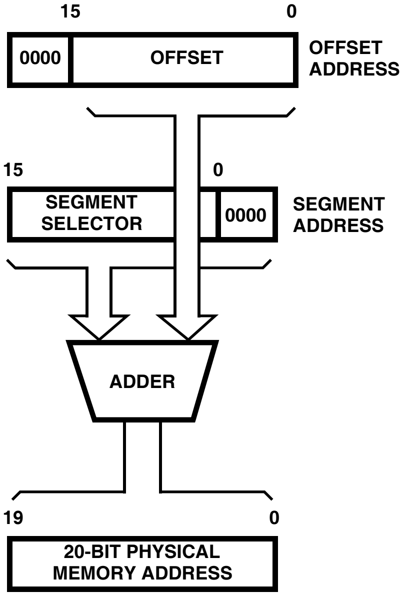
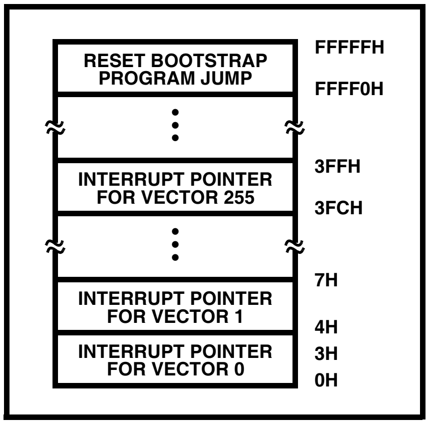


FIGURE 6. 80C286 REAL ADDRESS MODE ADDRESS CALCULATION

Reserved Memory Locations

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


RESET BOOTSTRAP PROGRAM JUMP
FFFFFH
FFFF0H
INTERRUPT POINTER FOR VECTOR 255
3FFH
3FCH
7H
INTERRUPT POINTER FOR VECTOR 1
4H
INTERRUPT POINTER FOR VECTOR 0
3H
0H

INITIAL CS:IP VALUE IS F000:FFF0

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

Interrupts

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

Protected Mode Initialization

To prepare the 80C286 for protected mode, the LIDT instruction is used to load the 24-bit interrupt table base and 16-bit limit for the protected mode interrupt table. This instruction can also set a base and limit for the interrupt vector table in real address mode. After reset, the interrupt table base is initialized to 000000(H) and its size set to 03FF(H). These values are compatible with 80C86 and 80C88 software. LIDT should only be executed in preparation for protected mode.

Shutdown

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

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

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

Protected Virtual Address Mode

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

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

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

Memory Size

The protected mode 80C286 provides a 1 gigabyte virtual address space per task mapped into a 16 megabyte physical address space defined by the address pins A_{23}-A_0 and $\overline{BHE}$. 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.

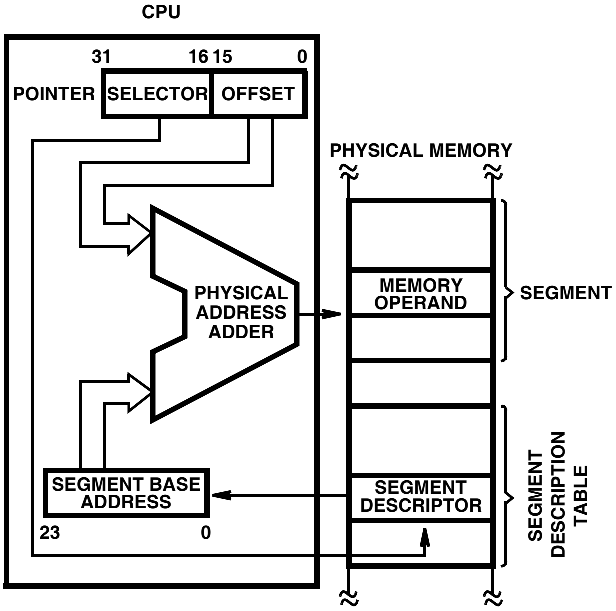


FIGURE 8. PROTECTED MODE MEMORY ADDRESSING

Memory Addressing

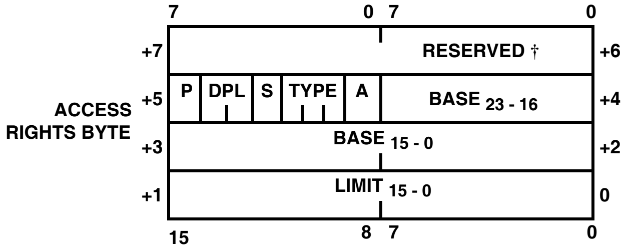
As in real address mode, protected mode uses 32-bit pointers, consisting of 16-bit selector and offset components. The selector, however, specifies an index into a memory resident table rather than the upper 16-bits of a real memory address. The 24-bit base address of the desired segment is obtained from the tables in memory. The 16-bit offset is added to the segment base address to form the physical address as shown in Figure 8. The tables are automatically referenced by the CPU whenever a segment register is loaded with a selector. All 80C286 instructions which load a segment register will reference the memory based tables without additional software. The memory based tables contain 8 byte values called descriptors.

Descriptors

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

Code and Data Segment Descriptors (S = 1)

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



† MUST BE SET TO 0 FOR COMPATIBILITY WITH FUTURE UPGRADES

FIGURE 9. CODE OR DATA SEGMENT DESCRIPTOR

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

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

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

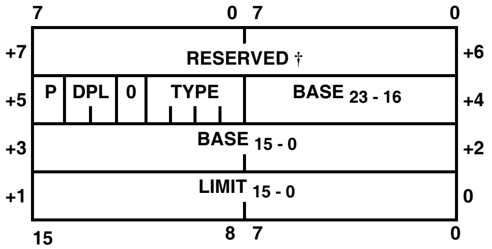
Data segments (S = 1, E = 0) may be either read-only or read-write as controlled by the W bit of the access rights byte. Read-only (W = 0) data segments may not be written into. Data segments may grow in two directions, as determined by the Expansion Direction (ED) bit: upwards (ED = 0) for data segments,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

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

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

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

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



† MUST BE SET TO 0 FOR COMPATIBILITY WITH FUTURE UPGRADES

FIGURE 10. SYSTEM SEGMENT DESCRIPTOR

TABLE 11. SYSTEM SEGMENT DESCRIPTOR FORMAT FIELDS

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

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

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

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

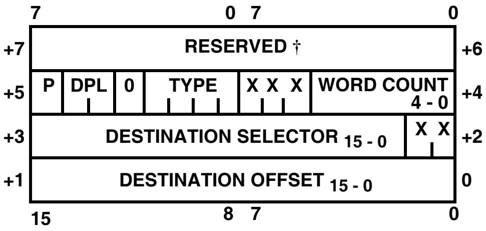
Exception 13 is generated when the gate is used if a destination selector does not refer to the correct descriptor type. The word count field is used in the call gate descriptor to indicate the number of parameters (0-31 words) to be automatically copied from the caller's stack to the stack of the called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

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

Segment Descriptor Cache Registers

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

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



† MUST BE SET TO 0 FOR COMPATIBILITY WITH FUTURE UPGRADES

FIGURE 11. GATE DESCRIPTOR

TABLE 12. GATE DESCRIPTOR FORMAT FIELD

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

Selector Fields

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

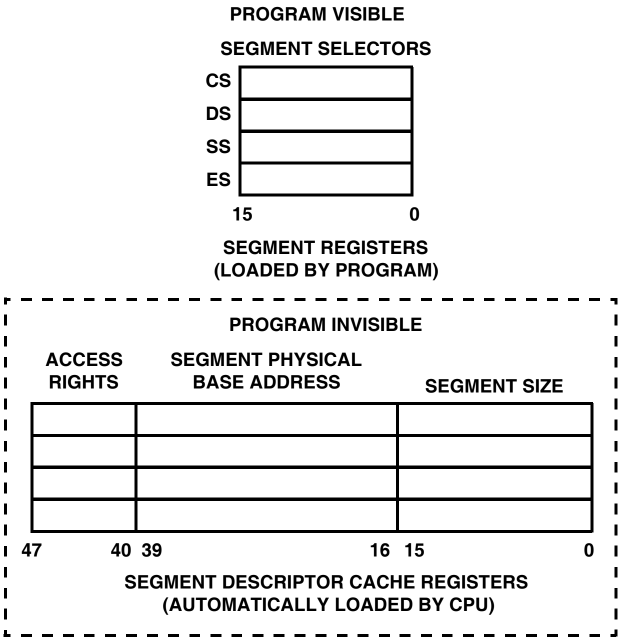
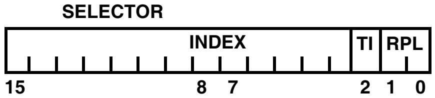


FIGURE 12. DESCRIPTOR CACHE REGISTERS



BITS	NAME	FUNCTION
1 - 0	Requested Privilege Level (RPL)	Indicates Selector Privilege Level Desired

2	Table Indicator (TI)	TI = 0 Use Global Descriptor Table (GDT)
		TI = 1 Use Local Descriptor Table (LDT)

15 - 3	Index	Select Descriptor Entry In Table

FIGURE 13. SELECTOR FIELDS

Local and Global Descriptor Tables

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

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

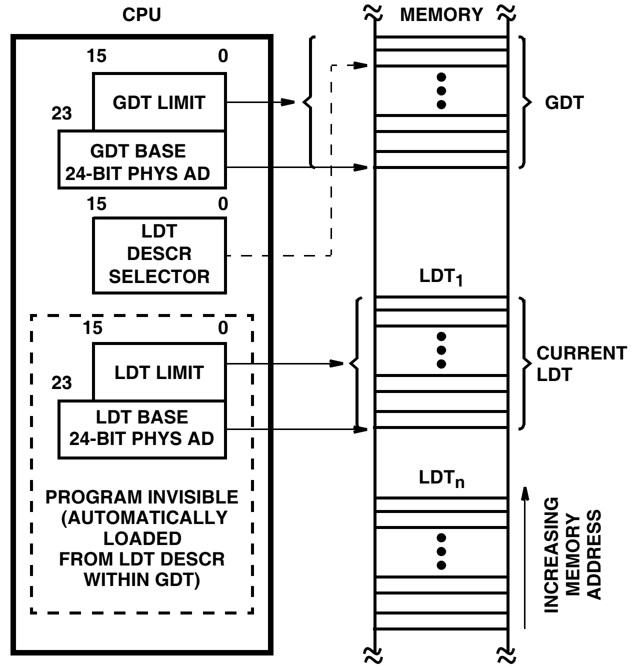


FIGURE 14. LOCAL AND GLOBAL DESCRIPTOR TABLE DEFINITION

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

	7 0	7 0	
+5	RESERVED †	BASE $_{23-16}$	+4
+3	BASE $_{15-0}$		+2
+1	LIMIT $_{15-0}$		0
	15 8	7 0	

† MUST BE SET TO 0 FOR COMPATIBILITY WITH FUTURE UPGRADES

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

Interrupt Descriptor Table

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

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

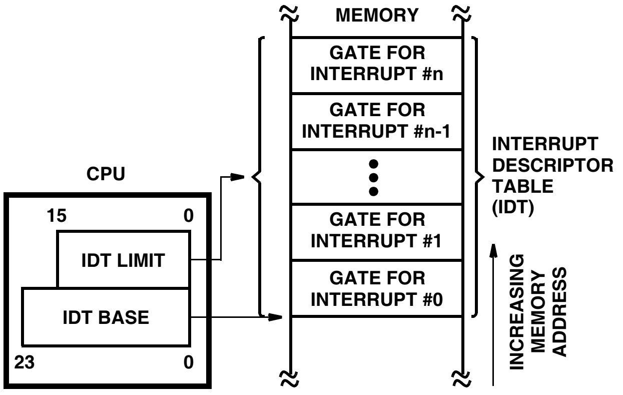
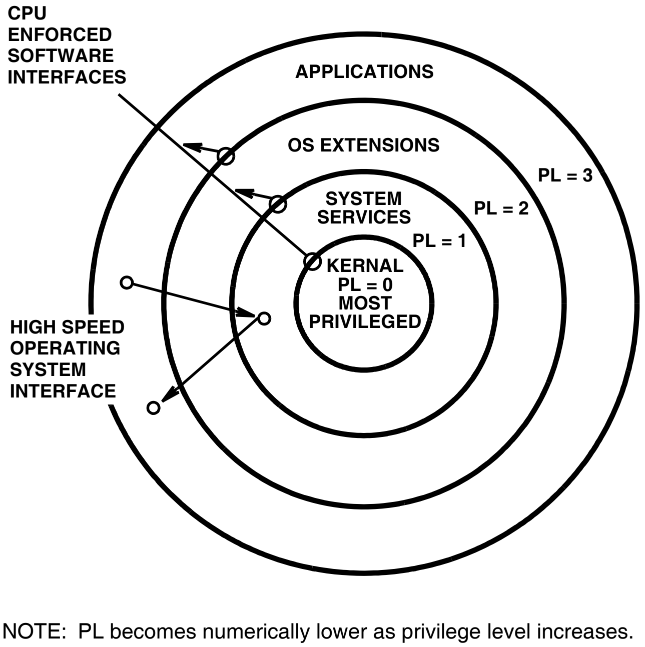


FIGURE 16. INTERRUPT DESCRIPTOR TABLE DEFINITION

Privilege

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

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



NOTE: PL becomes numerically lower as privilege level increases.

FIGURE 17. HIERARCHICAL PRIVILEGE LEVELS

Task Privilege

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

Descriptor Privilege

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

Selector Privilege

Selector privilege is specified by the Requested Privilege Level (RPL) field in the least significant two bits of a selector. Selector RPL may establish a less trusted privilege level than the current privilege level for the use of a selector. This level is called the task's effective privilege level (EPL). RPL can only reduce the scope of a task's access to data with this selector. A task's effective privilege is the numeric maximum of RPL and CPL. A selector with RPL = 0 imposes no additional restriction on its use while a selector with RPL = 3 can only refer to segments at privilege Level 3 regardless of the task's CPL. RPL is generally used to verify that pointer parameters passed to a more trusted procedure are not allowed to use data at a more privileged level than the caller (refer to pointer testing instructions).

Descriptor Access and Privilege Validation

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

Data Segment Access

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

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

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

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

TABLE 13. DESCRIPTOR TYPES USED FOR CONTROL TRANSFER

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

NOTES:

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

Control Transfer

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

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

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

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

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

The privilege rules on control transfer require:

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

Privilege Level Changes

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

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

Protection

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

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

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

TABLE 14. SEGMENT REGISTER LOAD CHECKS

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

TABLE 15. OPERAND REFERENCE CHECKS

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

NOTE: Carry out in offset calculations is ignored.

TABLE 16. PRIVILEGED INSTRUCTION CHECKS

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

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

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

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

Exceptions

The 80C286 detects several types of exceptions and interrupts in protected mode (see Table 17). Most are restartable after the exceptional condition is removed. Interrupt handlers for most exceptions can read an error code, pushed on the stack after the return address, that identifies the selector involved (0 if none). The return address normally points to the failing instruction including all leading prefixes. For a processor extension segment overrun exception, the return address will not point at the ESC instruction that caused the exception; however, the processor extension registers may contain the address of the failing instruction.

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

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

SPECIAL OPERATIONS

Task Switch Operation

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

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

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

TABLE 17. PROTECTED MODE EXCEPTIONS

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

NOTES:

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

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

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

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

Processor Extension Context Switching

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

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

Pointer Testing Instructions

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

Double Fault and Shutdown

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

Protected Mode Initialization

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

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

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

TABLE 18. 80C286 POINTER TEST INSTRUCTIONS

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