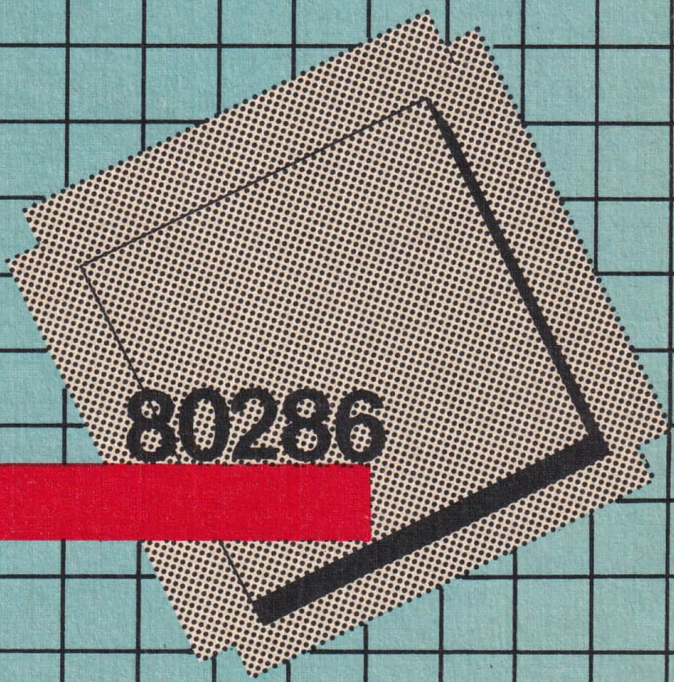
Technische Informatik

Claßen · Wiesner

Wissensspeicher 80286-Programmierung



VEB Verlag Technik Berlin

Claßen · Wiesner

Wissensspeicher 80286-Programmierung

Claßen, Ludwig:
Wissensspeicher 80286-Programmierung / Ludwig
Claßen; Ulrich Wiesner. - 240 S. : 41 Bilder, 12
Taf. - (Technische Informatik)
ISBN 3-341-00867-5
NE: Wiesner, Ulrich

ISBN 3-341-00867-5
ISSN 0863-0860

1. Auflage

VT 201 · 3/6051-1(83)
Printed in the German Demokratic Republic
Druck und buchbinderische Verarbeitung: Druckzentrum Berlin,
Grafischer Großbetrieb
Lektor: Jürgen Reichenbach
Einbandgestaltung: Gabriele Schwesinger
LSV 3054
Bestellnummer: 554 280 7
02400

Vorwort

Die Publikation von Grundlagenliteratur für die breite Nutzung der Mikroprozessortechnik hat im VEB Verlag Technik eine lange Tradition, die bis in das Jahr 1981 zurückreicht.
Solche Veröffentlichungen tragen im hohen Maß dazu bei, die angestrebte volkswirtschaftliche Effektivität beim Einsatz dieser Hochtechnologiebauelemente zu erreichen; denn was nutzt einem Anwender der beste Mikroprozessor oder Mikrocomputer, wenn er nicht weiß, welche Architekturmerkmale er hat oder wie er zu programmieren ist.

Seit 1986 hat sich besonders die Fachbuchreihe TECHNISCHE INFORMATIK das Ziel gestellt, möglichst aktuell und umfassend auch über neue Mikroprozessoren zu informieren. Hier sei an die Veröffentlichungen zum 8-Bit-Mikroprozessor U880, zur Einchip-Mikrocomputerfamilie U881 und zu den beiden 16-Bit-Mikroprozessorsystemen U8000 bzw. K1810WM86 erinnert.
Dieser Tradition entsprechend, soll mit dem vorliegenden "Wissensspeicher 80286-Programmierung" ein Lehr- und Nachschlagebuch zur Programmierung dieses modernen Schaltkreises bereitgestellt werden.

Der Mikroprozessor 80286 stellt international ein außerordentlich bedeutsames Bauelement dar. Das gilt, neben den technischen Leistungsparametern dieses Schaltkreises, auch für die wichtigen Fragen der Softwarekompatibilität unter den Betriebsystemen MS-DOS, OS/2 und UNIX.

Der vorliegende Band der Fachbuchreihe TECHNISCHE INFORMATIK behandelt den Aufbau, die interne Struktur, die verschiedenen Betriebsarten und natürlich den ganzen Komplex der Programmierung des 80286 aus dem Blickwinkel der Software.
Nach der Einleitung und einem Systemüberblick, in dem Fragen der Zuordnung des 80286 zu bekannten Bauelementefamilien und interne Architekturmerkmale des 80286 besprochen werden, behandelt der dritte und vierte Abschnitt die beiden grundverschiedenen Betriebsarten (real mode/protected mode) und den umfangreichen Registersatz dieses Schaltkreises.
Die Abschnitte 5. bis 7. vermitteln alle zur Speichernutzung wichtigen Informationen (Adreßraum, Speicherverwaltung, Zugriffsschutz).
Im achten Abschnitt wird das außerordentlich wichtige, von der Hardware des 80286 voll unterstützte Taskkonzept behandelt. Daran schließen sich im neunten und zehnten Abschnitt alle wichtigen Informationen zum Interruptsystem und zur peripheriebezogenen Ein-/Ausgabe an.
Der elfte, zwölfte und dreizehnte Abschnitt (Datenformate, Adressierungsarten, Befehlssatz) sind für den Programmierer ständiges Arbeitsmittel bei der Notation seiner Programmkodesequenzen.

In den beiden recht eigenständigen Abschnitten 14. und 15. schließlich werden wichtige Fragen der Systeminitialisierung und der Kompatibilität behandelt.

Zum Nachschlagen für den erfahrenen Programmierer sind die in den Anlagen enthaltenen Übersichtsinformationen (Befehlslisten, Kodierungsübersichten, Initialisierungstabellen usw.) vorgesehen.

Natürlich war es nicht möglich, alle Informationen über dieses oder jenes Detail bzw. über diese oder jene Kombinationsmöglichkeit von Befehlen und Betriebsarten des hochkomplexen 80286 auf den 240 Druckseiten des vorliegenden Bandes zusammenzufassen.
Die Autoren haben sich deshalb bemüht, das wirklich Wesentliche vom unwesentlichen Sonderfall zu trennen.
Ein umfangreiches Literaturverzeichnis soll gegebenenfalls beim Nachschlagen helfen. Im Zweifelsfall sind sicher immer experimentelle Untersuchungen notwendig.

Der Band ist gut zur Einarbeitung in die Konzepte des 80286 geeignet und für die Aus- und Weiterbildung an den Universitäten, Hoch- und Fachschulen verwendbar. Dabei werden Grundkenntnisse der Programmierung vorausgesetzt. Für den erfahrenen Programmierer soll der Band den Charakter eines Nachschlagewerks mit hoher Informationsdichte haben.

Trotz großer Sorgfalt bei der Manuskripterarbeitung ist es bei der Fülle des Stoffes nicht vollständig auszuschließen, daß sich Druck- oder Sachfehler eingeschlichen haben. Die Autoren sind deshalb jederzeit für Hinweise dankbar. Diese sind an den VEB Verlag Technik oder direkt an die Autoren im

Zentrum für Forschung und Technologie
des KEAW "Friedrich Ebert"
Storkower Str. 101
BERLIN
1055

zu richten.

Besonderen Dank gilt dem Herausgeber, Herrn Dr. G. Paulin, für viele wertvolle Hinweise, dem verantwortlichen Lektor des VEB Verlag Technik, Herrn J. Reichenbach, für die außerordentlich kooperative Zusammenarbeit sowie all den Fachkollegen des ZFT-KEAW, die durch vielfältige Anregungen, durch Diskussionen und durch gemeinsame Arbeit am Zustandekommen dieses Bandes ihren Anteil haben.

Berlin

Ludwig Claßen Ulrich Wiesner

INHALTSVERZEICHNIS

1. Einleitung

Die Entwicklung der Mikroprozessortechnik steht, obwohl sie nur ein Teilgebiet der modernen Mikroelektronik darstellt, in vieler Hinsicht im Mittelpunkt der Aufmerksamkeit einer breiten potentiellen Anwenderschicht. Von den ersten Mikroprozessorbauelementen Anfang der siebziger Jahre bis zu dem internationalen Industriestandard '80286' ist dabei eine in atemberaubendem Tempo vorangetriebene Entwicklung abgelaufen /1/ bis /5/.
Eckpunkte dieser Entwicklung und damit zugleich Stammväter des 80286 sind solche Bauelemente wie der 8-Bit-Mikroprozessor 8008, die 8-Bit-Mikroprozessoren 8080 bzw. Z80, besonders aber der 16-Bit-Mikroprozessor 8086 /6/ bis /8/.
Sie alle haben in der einen oder anderen Form Merkmale oder Eigenschaften hinterlassen, die, neben innovativen Ideen, heute im 16-Bit-Mikroprozessor 80286 wiederzufinden sind. Das betrifft den Aufbau und die Struktur des Registersatzes, der Befehlsliste, des Interruptsystems u.a.m.
Besonders hervorgehoben werden muß in diesem Zusammenhang die fast vollständige Software-Aufwärtskompatibilität zwischen den beiden Bauelementen 8086 und 80286. Diese Eigenschaft sichert die Lauffähigkeit der für einen 8086 geschriebenen Programme auch auf dem, dann allerdings mit gebremster Leistung arbeitenden 80286.
Diese Software-Aufwärtskompatibilität des 80286 zusammen mit der Tatsache, daß der international wohl größte Computerproduzent (IBM) vor vielen Jahren erst den 8086, später dann folgerichtig den 80286 und seine Nachfolger als Basis ihrer Personalcomputerlinie (XT, AT, PS/2) auswählte, machte den 80286 weltweit zu dem Industriestandard, den er heute darstellt.
Genannt werden müssen in diesem Zusammenhang unbedingt auch die Betriebssysteme MS-DOS und OS/2, deren Lauffähigkeit eben nur auf Mikroprozessoren dieser Familie gegeben ist /9/ /10/.
Natürlich wurde und wird dieses in der Vergangenheit so erfolgreiche Konzept der Software-Aufwärtskompatibilität in der Gegenwart und Zukunft weitergeführt, so daß international eine Mikroprozessor-Modellreihe

8086 80286 80386 80486

im Entstehen ist.
Sie bietet bei strenger Wahrung des Gedankens der Aufwärtskompatibilität und, angepaßt an die inzwischen erreichten Technologiefortschritte, für unterschiedliche Aufgabengebiete bei unterschiedlichen Kosten jeweils die entsprechende Mikroprozessorleistung.
An dieser Stelle muß auch der Name der Firma INTEL erwähnt werden, die diese Modellreihe konzipiert und weltweit durchgesetzt hat, obwohl heute mehrere Hersteller mit verschiedenen Technologien (NMOS, CMOS) zu einem oder mehreren dieser Bauelemente kompatible Mikroprozessoren herstellen /11/ bis /13/.

2. Systemübersicht

Moderne Mikroprozessorsysteme, wie der 80286, übernehmen heute Aufgaben aus den unterschiedlichsten Anwendungsbereichen. Das geht vom dezidierten Echtzeitsystem über den arbeitsplatzbezogenen Personalcomputer bis hin zur universellen mehrplatzfähigen UNIX-Workstation.
Die wichtigsten Eigenschaften des Mikroprozessors 80286, die diesen Anforderungen entsprechen müssen, lassen sich in wenigen Punkten zusammenfassen:

- echter 16-Bit-Mikroprozessor mit 16-Bit-Datenbus und 24-Bit-Adreßbus,
- Taktfrequenz je nach Spezifikation 8, 12, 16, 20 MHz,
- zwei alternative Betriebsarten:
  REAL ADDRESS MODE (8086-Betriebsart) mit bis zu 1 MByte physisch adressierbarem Speicher,
  PROTECTED VIRTUAL ADDRESS MODE (80286-Betriebsart) mit bis zu 16 MByte physisch adressierbarem Speicher,
- integrierte Speicherverwaltungseinheit MMU (memory management unit),
- vier unterschiedliche Privilegierungsstufen zur Sicherung des Zugriffsschutzes,
- hardwaregestützte Taskwechselmechanismen mit automatischer Statussicherung,
- 256 unterschiedliche vektorisierte Interruptebenen,
- leistungsfähiger Befehlssatz mit Multiplikation und Division,
- Kommunikationsmöglichkeit mit Coprozessoren.

Der 16-Bit-Mikroprozessor 80286 besitzt eine interne Struktur, die aus vier parallel zueinander arbeitenden, weitestgehend eigenständigen Einheiten aufgebaut ist (Bild 2.1):

BUSEINHEIT (bus unit) - Die Buseinheit dient der Steuerung und Überwachung der Zugriffe des 80286 auf den Systemspeicher und auf die Ein-/Ausgabe-Kanäle. Die Buseinheit holt automatisch bis zu sechs Befehlsbytes in einen internen Cachespeicher, der als Programmkode-Warteschlangenspeicher für die nächsten auszuführenden Befehle fungiert. Dadurch ist eine wesentliche Erhöhung des Befehlsdurchsatzes, auch bei relativ langsamen Speicherbauelementen, möglich.
Die Buseinheit wird auch bei der Zusammenarbeit mit einem externen Coprozessor aktiv.

BEFEHLSEINHEIT (instruction unit) - Die Befehlseinheit des 80286 übernimmt jeweils alle zu einer Instruktion gehörenden Befehlsbytes aus dem Programmkode-Warteschlangenspeicher der Buseinheit und unterzieht die Befehle einer internen Vorverarbeitung (Direktoperanden, Mikrokodeauswahl usw.). In der Befehlseinheit wird außerdem eine drei Befehle umfassende Befehlswarteschlange (instruction queue) aufgebaut.

AUSFÜHRUNGSEINHEIT (execution unit) - Die Ausführungseinheit übernimmt die komplette Befehlsausführung einer 80286-Instruktion mit Hilfe der in einem internen Mikrokode-ROM abgelegten Mikrobefehle (ROM read only memory).

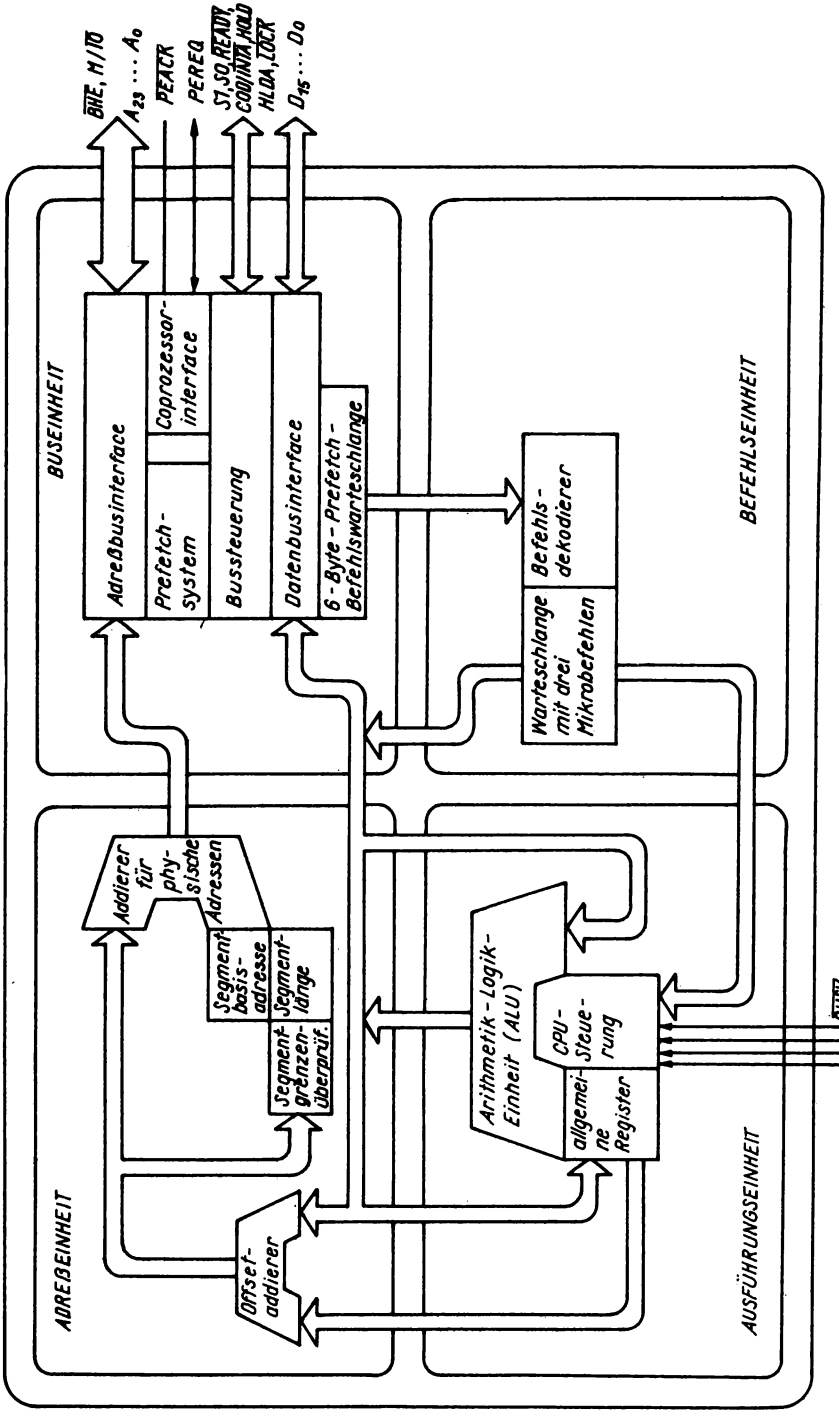


Bild 2.1. Interne Struktur des Mikroprozessors 80286

Die Ausführungseinheit enthält die Allzweckregister und die ALU - das eigentliche Rechenwerk des Mikroprozessors 80286 (ALU arithmetic logic unit).
Die Ausführungseinheit kann in der Befehlsabarbeitung durch externe Interruptsignale und durch Statussignale eines möglichen externen Coprozessors in ihrer Arbeit beeinflußt werden.

ADREßEINHEIT (address unit) - Die Adreßeinheit übernimmt die Berechnung der anzusprechenden physischen Speicheradressen aus der Segmentbasisadresse, dem Offsetanteil und einem ggf. vorhandenen Distanzwert.
Die Adreßeinheit zeichnet auch verantwortlich für alle Zugriffsschutzüberprüfungen. Sie enthält die für diese Berechnungen und Überprüfungen notwendigen Adreßregister des 80286.

Mit seiner Umwelt ist der 80286 über insgesamt 68 Anschlußpunkte seines als 'pin-grid-array' bezeichneten quadratischen Gehäuses verbunden (Bild 2.2).
Neben dem bidirektionalen 16-Bit-Datenbus und dem unidirektionalen 24-Bit-Adreßbus existieren sechs Steuersignale für die Prozessorzustandsanzeige sowie zur Buszyklussteuerung, zwei Interruptsignaleingänge, vier Signale, die die Zusammenarbeit des 80286 mit einem Coprozessor steuern und überwachen, ein Systemresetsignaleingang, ein Takteingang und schließlich drei Anschlüsse, die der Versorgungsspannungszuführung sowie einer Kondensatorverbindung (Ableitung von Spannungsspitzen) vorbehalten sind.
Die genaue Arbeitsweise der 80286-Signalanschlußpins ist entsprechenden Hardwareunterlagen zu entnehmen /12/ bis /14/.
Neben dem eigentlichen Mikroprozessor 80286 existieren vier der unmittelbaren Umgebung dieses CPU-Bausteins (CPU central processing unit) zugeordneten Standardchips:

- 82284 phasengleicher Taktgenerator für den 80286,
- 82288 Buscontroller zur Erzeugung der Ansteuersignale für Speicher und Ein-/Ausgabe aus den Statussignalen des 80286,
- 82289 Busarbiter zum Anschluß des 80286 an ein Multibus-I-System /21/ /22/,
- 80287 Numerikcoprozessor für schnelle, hochgenaue arithmetische Berechnungen /23/.

Diese mehr oder weniger komplexen Bausteine können, müssen aber nicht in einem 80286-Mikrocomputersystem Verwendung finden. Ihre Funktion kann, wenn günstig oder notwendig, auch von andernen Chips übernommen oder weggelassen werden.
Periphere Ein-/Ausgabe-Bausteine mit universellem Charakter zur parallelen oder seriellen Datenkommunikation, für Zähler/Zeitgeber-Aufgaben, zur Interruptsteuerung, für DMA-Aufgaben (DMA direct memory access) u.a.m. können selbstverständlich, je nach Notwendigkeit oder Verfügbarkeit, mit dem 80286 zusammenarbeiten. Hier sei auf die einschlägigen Datenbücher verwiesen.

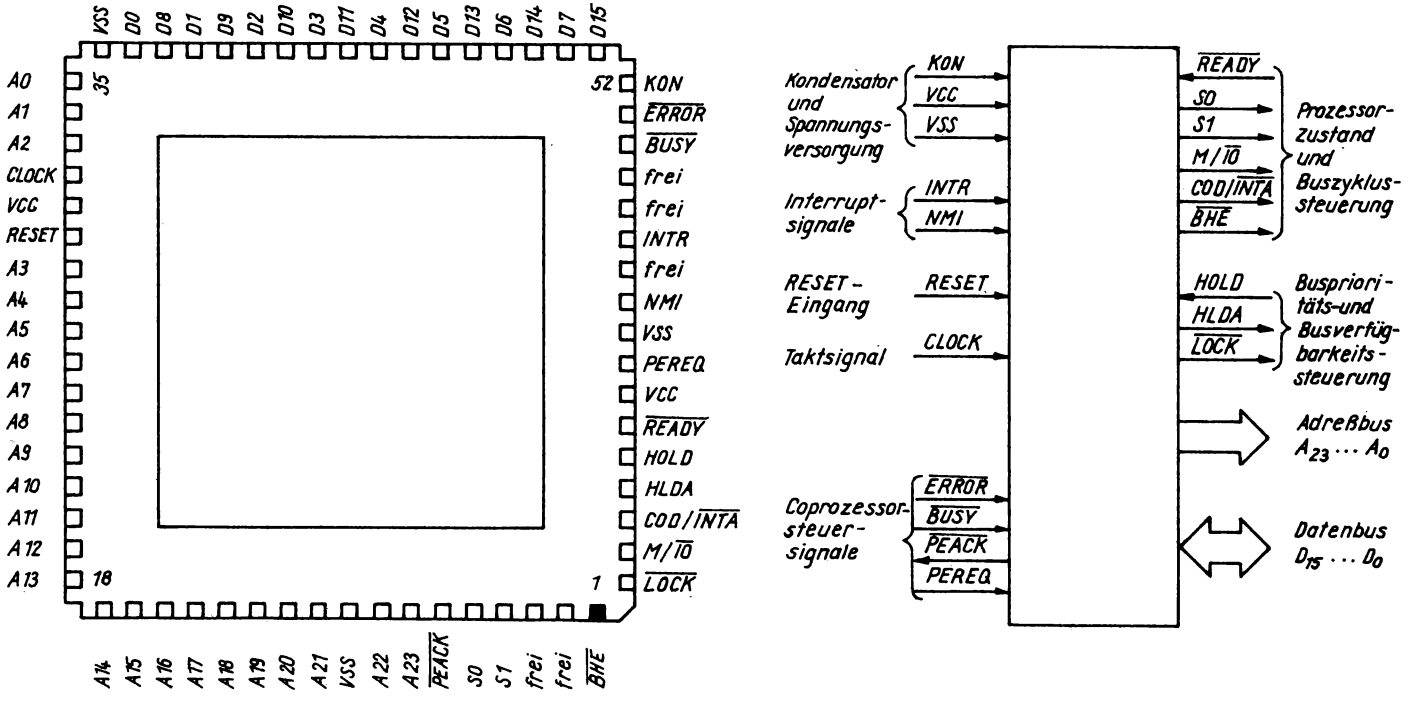


Bild 2.2. Physische und funktionsbezogene Signalbelegung des Mikroprozessors 80286

3. Betriebsarten

Der Mikroprozessor 80286 kann alternativ in zwei grundsätzlich unterschiedlichen Betriebsarten arbeiten:

REAL ADDRESS MODE

PROTECTED VIRTUAL ADDRESS MODE.

Die Auswahl einer der beiden möglichen Betriebsarten erfolgt softwaregesteuert vom Programmierer. Sie unterscheiden sich besonders durch stark voneinander abweichende Adreßräume, unterschiedliche Speicherverwaltungsstrukturen, die Verfügbarkeit hardwaregestützter Zugriffsschutzfunktionen, Taskwechselmechanismen u.a.m.
Die Betriebsart REAL ADDRESS MODE hat ihre besondere Bedeutung bei der Sicherung der Aufwärtskompatibilität auf Maschinenkodeebene zwischen allen älteren 8086-kompatiblen Mikroprozessorfamilien und dem 80286.
Die Betriebsart PROTECTED VIRTUAL ADDRESS MODE ermöglicht die Ausnutzung aller modernen innovativen Leistungsmerkmale des Mikroprozessors 80286 durch den Anwender.

3.1. Betriebsart REAL ADDRESS MODE (RA-Mode)

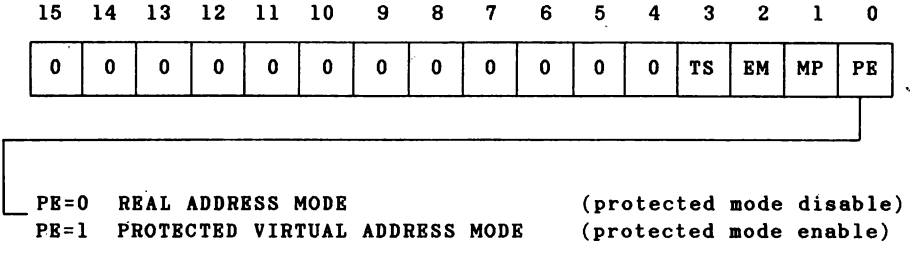
Nach jedem Netzeinschalt- oder Systemreset befindet sich der Mikroprozessor 80286 zunächst immer in der Betriebsart REAL ADDRESS MODE.
Die Betriebsart REAL ADDRESS MODE des Mikroprozessors 80286 ist besonders dadurch gekennzeichnet, daß alle in einem Programm verwendeten Adressen reale physische Adressen darstellen, die im direkten Bezug zu Hardwareadressen im Systemspeicher stehen.
Die Größe des adressierbaren Speicherbereichs beträgt insgesamt 1 MByte.
Die in den Abschnitten 7. und 8. zu behandelnden hardwaregestützten Zugriffsschutz- und Taskwechselmechanismen des 80286 stehen in der Betriebsart REAL ADDRESS MODE dem Programmierer nicht zur Verfügung. In allen anderen Abschnitten dieses Buches werden die Unterschiede der beiden 80286-Betriebsarten gesondert ausgewiesen.
Wichtig ist, an dieser Stelle anzumerken, daß der Mikroprozessor 80286 in der Betriebsart REAL ADDRESS MODE eine weitestgehende Architektur- und Maschinenkodekompatibilität mit allen 8086-kompatiblen Mikroprozessorsystemen besitzt (Abschn. 15. Kompatibilität).
Für die Bezeichnung 'REAL ADDRESS MODE' soll im folgenden auch die Abkürzung 'RA-Mode' Verwendung finden.

3.2. Betriebsart PROTECTED VIRTUAL ADDRESS MODE (PVA-Mode)

Der PROTECTED VIRTUAL ADDRESS MODE verbindet die Möglichkeiten des REAL ADDRESS MODE mit Speicherverwaltungsfunktionen (memory management), Speicherverletzungsschutzfunktionen (protection violation), der Hardwareunterstützung von Multi-Task-Systemen, der Realisierung virtueller Adressierungsmechanismen und weiterer Eigenschaften, die alle der Implementierung moderner Softwaresystemstrukturen dienen.

Die Betriebsart PROTECTED VIRTUAL ADDRESS MODE des Mikroprozessors 80286 ist besonders dadurch gekennzeichnet, daß alle in einem Programm verwendeten Adressen virtuellen Charakter haben und über eine Speicherverwaltungseinheit in der 80286-CPU in physische Adressen umgewandelt werden. Der direkte Zugriff zu physischen Adressen im Speicher des 80286-Mikrorechners wird nicht gestattet.
Der gesamte verfügbare Speicher, sowohl der primäre Hauptspeicher als auch der sekundäre Massenspeicher, wird als ein großer, sogenannter virtueller Speicherraum betrachtet, der insgesamt 1 Gigabyte pro Task umfaßt (Abschnitte 5. Adreßraum und 8. Taskkonzept).
Nach dem Systemreset befindet sich der Mikroprozessor 80286 zunächst grundsätzlich, wie bereits erwähnt, in der Betriebsart REAL ADDRESS MODE. Der Übergang vom REAL ADDRESS MODE in den PROTECTED VIRTUAL ADDRESS MODE erfolgt durch Setzen des PE-Bits (protected mode enable) im Maschinenstatuswort (MSW) des 80286 (Abschn. 4.5. Maschinenstatuswort).
Ein Rücksetzen des PE-Bits ist nur durch ein erneutes Netzeinschalt- oder Systemresetsignal möglich, so daß ein programmgesteuerter Übergang vom PROTECTED VIRTUAL ADDRESS MODE in den REAL ADDRESS MODE prinzipiell ausgeschlossen wird.
Vom Anwender des Mikroprozessors 80286 ist also eine der beiden Betriebsarten für eine konkrete Systemkonfiguration generell auszuwählen.

Maschinenstatuswort des Mikroprozessors 80286:



Für die Bezeichnung 'PROTECTED VIRTUAL ADDRESS MODE' soll im folgenden auch die Abkürzung 'PVA-Mode' Verwendung finden.

4. Registersatz

Der Mikroprozessor 80286 enthält insgesamt 19 Einzelregister und Registergruppen, von denen einige ausschließlich für die Softwaresystemkonfiguration genutzt werden. Sie stehen damit dem Programmierer bei der Applikationsprogrammerarbeitung nicht zur Verfügung.
Die Einteilung der Register des Mikroprozessors 80286 erfolgt in die Gruppen

allgemeinen Register,
Segmentregister,
Systemkonfigurationsregister

sowie in drei Einzelregister (FLAGS, MSW, IP).
Die drei Einzelregister (FLAGS, MSW, IP) dienen zur Statusanzeige und für Steuerinformationen (Bild 4.1).
Bei der Behandlung des Registersatzes des Mikroprozessors 80286 ist unbedingt zu beachten, daß, abhängig von der eingestellten Betriebsart (RA-Mode oder PVA-Mode), zwei unterschiedliche Segmentregisterstrukturen existieren.
Die Systemkonfigurationsregister existieren grundsätzlich nur im PVA-Mode.

4.1. Allgemeine Register

Die acht allgemeinen Register des Mikroprozessors 80286 sind jedem Programmierer voll zugänglich. Jeweils 16 Bit breit, dienen sie primär der temporären Abspeicherung von Daten und Adressen, die bei der Programmabarbeitung entstehen - Operanden arithmetischer oder logischer Operationen, Zählerwerte von Laufvariablen, Adressierungsindizes u.a.m.
Die Bezeichnung der allgemeinen Register (Registermnemoniks) ist dem Bild 4.2 zu entnehmen. Sie ist bei vier der acht allgemeinen Register davon abhängig, ob der volle 16-Bit-Wert adressiert werden soll (Bezeichnung: AX, BX, CX, DX) oder ob nur auf einen 8-Bit-Auswahlwert im 16-Bit-Register, jeweils für High- oder Low-Teil unterschiedlich spezifiziert, zugegriffen wird (Bezeichnung: AH, AL, BH, BL, CH, CL, DH, DL).
Der Bytezugriff auf vier der acht allgemeinen Register ist in verschiedenen Situationen für den Programmierer bei der Behandlung von bytestrukturierten Daten unumgänglich. Eine besondere Bedeutung hat er außerdem bei der Sicherung von Kompatibilitätseigenschaften (Abschn. 15. Kompatibilität).
Alle acht allgemeinen Register sind vom Programmierer prinzipiell freizügig nutzbar. Trotzdem sind bestimmten Registern des allgemeinen Registersatzes spezielle Eigenschaften oder Funktionen unterlegt, die in einem Programm zu einer zweckbestimmten Nutzung zwingen.
Im einzelnen sind das folgende Eigenschaften oder Funktionen:

AX, DX Operandenregister bei arithmetischen Operationen (insbesondere Addition, Subtraktion, Multiplikation, Division) und bei Ein-/Ausgabe-Operationen,

BX, BP Anfangs- oder Basisadreßregister von Datenstrukturen (Tabellen),

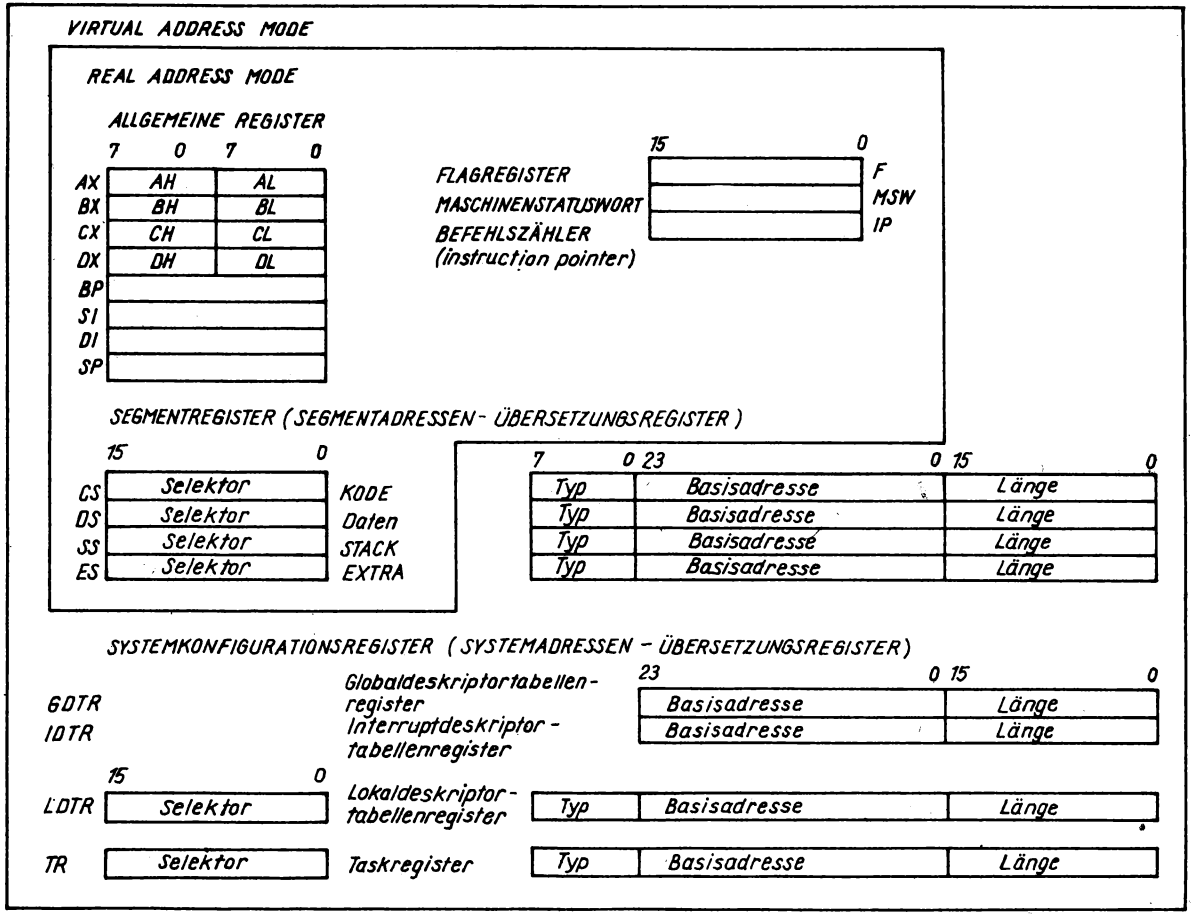


Bild 4.1. Registersatz des Mikroprozessors 80286

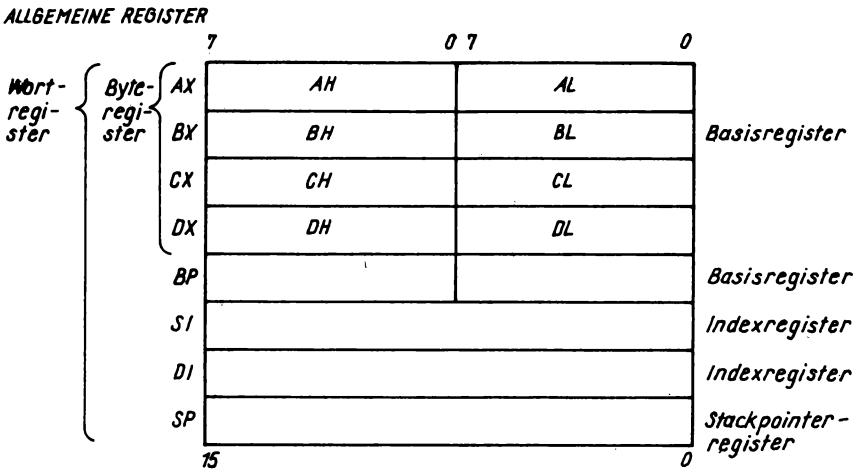


Bild 4.2. Allgemeine Register des Mikroprozessors 80286

CX Schleifenzählregister bei Wiederholungsbefehlen (repeat) und Operandenregister bei logischen Verschiebeoperationen (shift, loop),

SI, DI Indexregister bei der inkrementalen indizierten Adressierung (SI source index; DI destination index),

SP Stackpointerregister.

Aus dieser Aufzählung verschiedener Besonderheiten bestimmter allgemeiner Register hebt sich die Funktion der Indexregister (SI, DI), der Basisregister (BP, BX) und des Stackpointerregisters (SP) in ihrer Bedeutung deutlich ab. Sie ist noch genauer zu behandeln.
Dabei ist zu beachten, daß bei der Verwendung der allgemeinen Register SI, DI, BP, BX, SP als Index-, Basis- oder Stackpointerregister der jeweilige momentane Inhalt dieser Register immer einen vom Programmierer spezifizierten Distanzwert (offset) zur Anfangsadresse eines Speichersegments kennzeichnet.
Wie in den Abschnitten 4.2. (Segmentregister) und 5. (Adreßraum) genauer zu behandeln sein wird, existieren beim Mikroprozessor 80286 vier prinzipiell zu unterscheidende Speichersegmenträume für

Programmkodebereiche,
Datenbereiche (lokal),
Stackbereiche und
Extradatenbereiche (global).

Werden die Register SI, DI, BX, BP und SP - wie vorgesehen - als Index-, Basis- oder Stackpointerregister benutzt, lassen sich die daraus resultierenden Zugriffsverfahren jeweils nur auf Datenspeicher- bzw. Stackspeichersegmente anwenden.
Im einzelnen gilt dabei die folgende Zuordnung:

SI	Datensegmentzugriff,
DI	Datensegmentzugriff (bei Stringoperationen auch Zugriff auf das Extradatensegment),
BX	Datensegmentzugriff,
BP	Stacksegmentzugriff,
BX+SI, DI	Datensegmentzugriff,
BP+SI, DI	Stacksegmentzugriff,
SP	Stacksegmentzugriff.

Die beim Mikroprozessor 80286 verfügbaren Adressierungsverfahren werden im Abschnitt 12. genauer behandelt.

4.1.1. Indexregister

Die beiden Register SI (source index) und DI (destination index) können bei Bedarf als Indexregister verwendet werden. Sie spielen dann eine besondere Rolle bei dem indizierten Adressierungsverfahren des Mikroprozessors 80286.
Beiden Registern obliegt bei dieser Zugriffsmethode die Funktion, einen Distanzwert (displacement) vorzugeben, der die Differenz zwischen der Anfangsadresse des aktuellen Datenbereichs und der Speicherposition, auf die zugegriffen werden soll, spezifiziert.
Das indizierte Adressierungsverfahren wird von der Hardware des Mikroprozessors 80286 unterstützt (Abschn. 13. Befehlssatz). Es dient der besonders effektiven Adressierung in Datenstrukturen, bei denen ein Laufindex genutzt wird, um auf die einzelnen Datenelemente eines Feldes schnell und unkompliziert zugreifen zu können.
Bei Bedarf kann das indizierte Adressierungsverfahren auch mit dem im Anschluß zu behandelnden Basisadressierungsverfahren kombiniert angewendet werden.
Grundsätzlich ist allerdings zu beachten, daß mit Hilfe der indizierten Adressierung nur Speicherelemente in den oben angegebenen Segmenten erreicht werden können.

4.1.2. Basisregister

Die beiden Register BX und BP können bei Bedarf als Basisregister verwendet werden. Sie spielen dann eine besondere Rolle bei dem Basisadressierungsverfahren des Mikroprozessors 80286. Beiden Registern obliegt bei dieser Zugriffsmethode die Funktion, die Basisspeicheradresse einer Datenstruktur vorzugeben, innerhalb derer mit dem Basisadressierungsverfahren auf einzelne Datenelemente zugegriffen werden soll.

Das Basisadressierungsverfahren wird von der Hardware des Mikroprozessors 80286 umfangreich unterstützt (Abschn. 13. Befehlssatz). Es dient der besonders effektiven Adressierung in Datenstrukturen, bei denen, relativ zu einer bestimmten Basisadresse, auf die einzelnen Datenelemente der Struktur zugegriffen werden soll.
Wenn notwendig, kann das Basisadressierungsverfahren auch mit dem schon behandelten indizierten Adressierungsverfahren kombiniert angewendet werden.

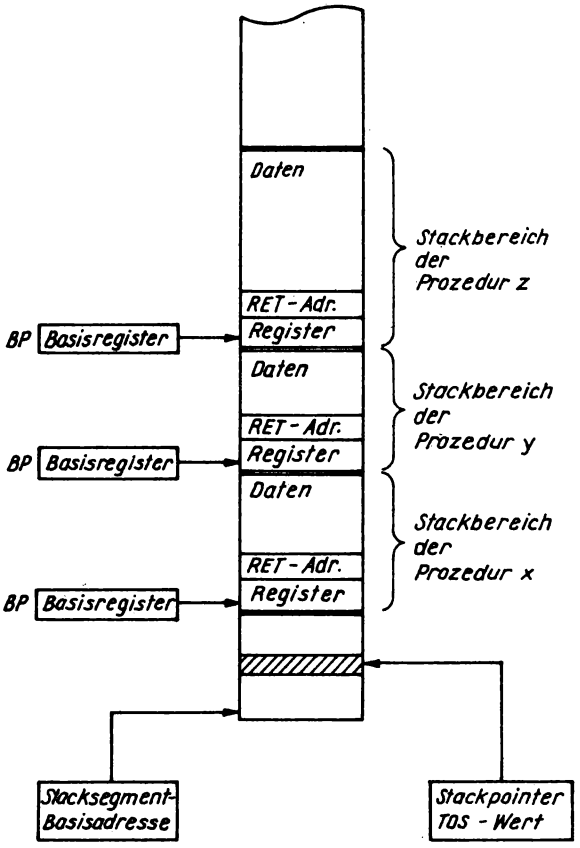


Bild 4.3. Nutzung des Registers BP als Basispointer des Stackrahmens mehrerer Prozeduren beim Mikroprozessor 80286

Genauso wie beim indizierten Adressierungsverfahren ist zu beachten, daß mit Hilfe der Basisadressierung nur Speicherelemente in den im Abschnitt 4.1. angegebenen Segmenten erreicht werden können. Das ist besonders beim Basisregister BP zu beachten, das dann nur den Zugriff im Stacksegment gestattet.

Ein wichtiges Anwendungsbeispiel für die Basisadressierung mit dem Register BP sind die im Zusammenhang mit der Stackorganisation definierbaren Basispointer des Stackrahmens (stack frame pointer).
Stackrahmen sollen den Zugriff auf im Stacksegment abgelegte Daten einer oder mehrerer Prozeduren, unabhängig von einem sich ggf. ändernden Stackpointerwert, ermöglichen (Bild 4.3). Diesen Basisadressierungszugriff innerhalb des Stack kann allerdings nur das Basisregister BP übernehmen.

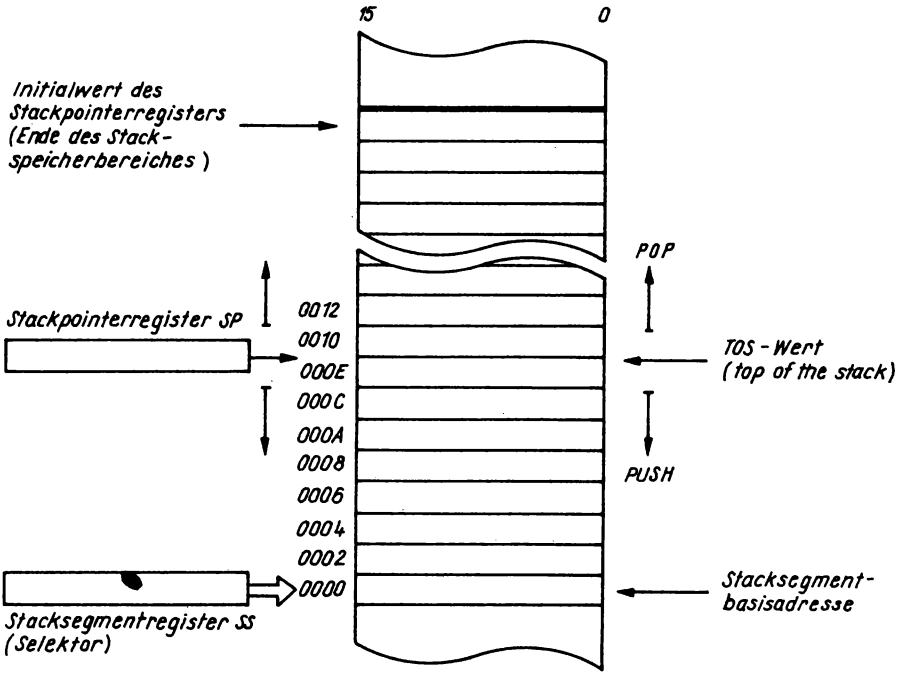


Bild 4.4. Stackorganisation beim Mikroprozessor 80286

4.1.3. Stackpointerregister

Der Inhalt des Stackpointerregisters SP kennzeichnet den jeweiligen Füllstand eines Stackdatenbereichs (Kellerspeicherbereich). Stackdatenbereiche werden in vom Programmierer festgelegten Stackspeichersegmenten angeordnet.
Die Nutzung eines Stackdatenbereichs erfolgt beim Mikroprozessor 80286, wie bei allen anderen Standardmikroprozessorsystemen, nach dem LIFO-Prinzip (LIFO last in first out).
Dieses Prinzip geht davon aus, daß die zuletzt eingespeicherten Daten als erste wieder ausgelesen werden.
Die Anzahl der maximal einzuspeichernden Werte hängt von der Größe des Stackdatenbereichs ab, der zur Aufnahme der Rückkehradressen bei Unter-

programmaufrufen bzw. Interruptserviceroutinen genauso dient wie zur Parameterübergabe oder zur Statusabspeicherung (Bild 4.4).
Die im Systemspeicher eines 80286-Mikrorechnersystems befindlichen Stackdatenbereiche werden durch den Inhalt des Stacksegmentregisters SS in ihrer Anordnung definiert (Abschn. 4.2. Segmentregister).
Der Inhalt des Stackpointerregisters SP spezifiziert dann den momentanen Füllstand des jeweiligen Stackdatenbereichs.
Mit SS:SP soll im folgenden die durch das Stacksegmentregister und durch das Stackpointerregister adressierte Speicheradresse gekennzeichnet werden.
Die Anzahl der vom Programmierer durch Laden des Stacksegmentregisters definierbaren Stackdatenbereiche ist nahezu unbegrenzt.
Zu einem bestimmten Zeitpunkt der Abarbeitung eines Programms ist allerdings nur immer ein durch den jeweiligen Inhalt des Stacksegmentregisters festgelegter, sogenannter momentaner Stackdatenbereich direkt aktiv.
Der zugehörige momentane Inhalt des Stackpointerregisters wird auch als TOS-Wert bezeichnet (TOS top of the stack).
Die Größe eines Stackdatenbereichs kann, entsprechend der 16-Bit-Adressierungsbreite des Stackpointerregisters SP, bis zu 64 KByte betragen.
Der Zugriff auf den Stackdatenbereich erfolgt grundsätzlich über 16-Bit-Worte, so daß das Einlagern (push) eines Wertes in den Stack mit einer zweifachen Dekrementierung des Inhalts des Stackpointerregisters verbunden ist, während das Auslagern (pop) eines im Stack abgelegten Wertes mit einer zweifachen Inkrementierung des TOS-Wertes einhergeht.

4.2. Segmentregister

Die Funktion der Segmentregister des Mikroprozessors 80286 ist eng mit Fragen der Adreßraum- und Speicherverwaltung in diesem System verbunden (Abschnitte 5. Adreßraum und 6. Speicherverwaltung).
Unter einem Segment wird eine logische Einheit eines in sich geschlossenen, nicht unterbrochenen Speicherbereichs verstanden.
Solche in sich geschlossenen, nicht unterbrochenen Segmente lassen sich in einem konkreten Softwaresystem eines Mikrorechners beispielsweise für

Programmkodeteile,
lokale oder globale Datenbereiche und
Stackbereiche

definieren.
Ein derartiges Speichermodell unterstützt die wohldurchdachte Strukturierung der Software und ist außerordentlich hilfreich bei der Nutzung gegenseitig wirkender Verriegelungsmechanismen (Bild 4.5).
Der Mikroprozessor 80286 besitzt intern vier Segmentregister, die als

Programmkodesegmentregister (CS),
Datensegmentregister (DS),
Stacksegmentregister (SS) und
Extradatensegmentregister (ES)

bezeichnet werden (Bild 4.6).

Sie dienen zur Auswahl von vier jeweils aktuellen Speichersegmenten, die zu einem bestimmten Zeitpunkt während der Abarbeitung eines Programms aktiv sind.
Dabei wird generell davon ausgegangen, daß ein in Abarbeitung befindliches Programm gleichzeitig auf vier unabhängig voneinander definierbare Speichersegmenträume in einem 80286-Mikroprozessorsystem zugreifen kann - auf ein Programmkodesegment (code), ein Stacksegment (stack) und zwei Datensegmente (data, extra data).
Der Inhalt eines Segmentregisters wird auch als Segmentselektor bezeichnet.
Bei der Diskussion der Segmentregister des Mikroprozessors 80286 ist zwischen den beiden möglichen Betriebsarten dieses Schaltkreises zu unterscheiden.

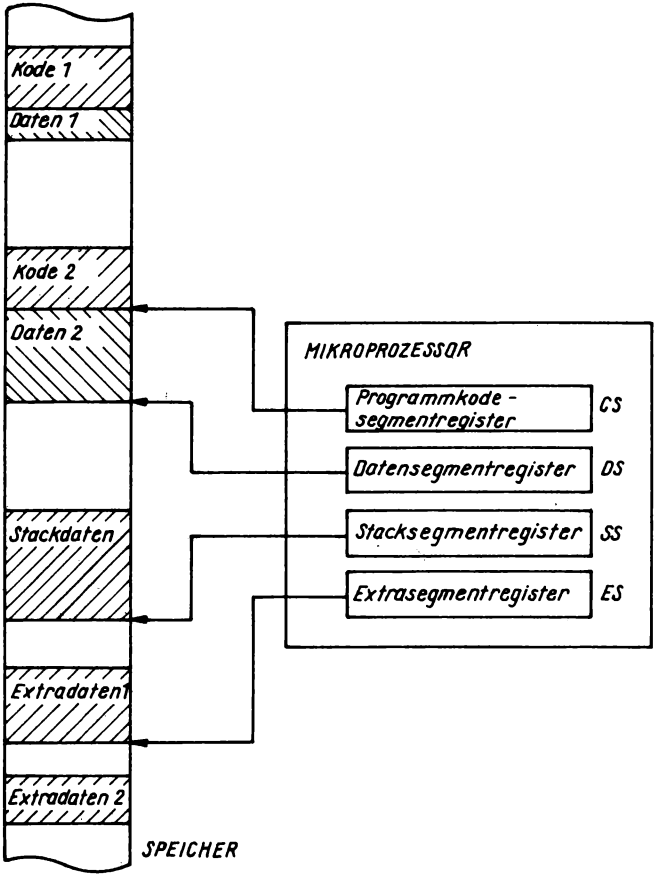


Bild 4.5. Speichersegmentierung beim Mikroprozessor 80286

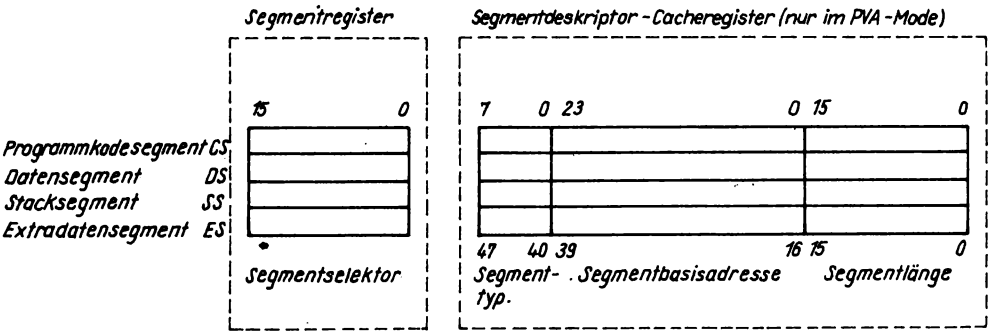


Bild 4.6. Segmentregister des Mikroprozessors 80286

In der Betriebsart REAL ADDRESS MODE beinhalten die vier 16-Bit-Segmentregister einen Segmentselektor, der den Versatz zwischen der physischen Speicheradresse 0 und der physischen Anfangsadresse eines ausgewählten 64 KByte großen Speichersegments im 1 MByte großen RA-Mode-Adreßraum (00000H - 0FFFFFH ; H Hexadezimalkennzeichen) des Mikroprozessors 80286 spezifiziert (Bild 4.7).
In der Betriebsart PROTECTED VIRTUAL ADDRESS MODE spezifiziert der Inhalt des 16-Bit-Segmentregisters nicht direkt eine physische Anfangsadresse eines Speichersegments, sondern wählt eines von 16384 (16 K) möglichen Segmenten im virtuellen Adreßraum einer Task aus.
Diese Auswahl erfolgt in mehreren, im Speicher des 80286-Mikroprozessorsystems vom Programmierer anzulegenden Segmentdeskriptortabellen, die alle ein Speichersegment spezifizierenden Informationen für die 16384 möglichen Segmente einer Task beinhalten (Bild 4.8).
Damit bei der Adreßbildung während der Programmabarbeitung nicht immer der Umweg über die Segmentdeskriptortabelle im Speicher des 80286-Mikrorechners gemacht werden muß, lädt der Mikroprozessor 80286 automatisch bei einer Veränderung eines der vier Segmentregister den gesamten Segmentdeskriptorinhalt des spezifizierten Segments in ein im Registersatz des Mikroprozessor 80286 befindliches Segmentdeskriptor-Cacheregister.
Das geschieht in einem laufenden Programm bei jedem Wechsel des Inhalts eines der vier Segmentregister, also immer beim Umschalten von CS, DS, SS oder ES auf ein anderes Speichersegment.
Nach diesem Umladen kostet dann die Adreßberechnung im PVA-Mode keine zusätzliche Prozessorzeit durch umständliches Nachschauen in den Segmentdeskriptortabellen im 80286-Systemspeicher.
Das sicher relativ selten vorkommende Neuladen des Segmentdeskriptor-Cacheregisters in einem laufenden Programm (z.B. bei der Abarbeitung von CALL- oder JMP-Befehlen) kann in der Gesamtrechenzeitbilanz eines 80286-Systems fast vernachlässigt werden.

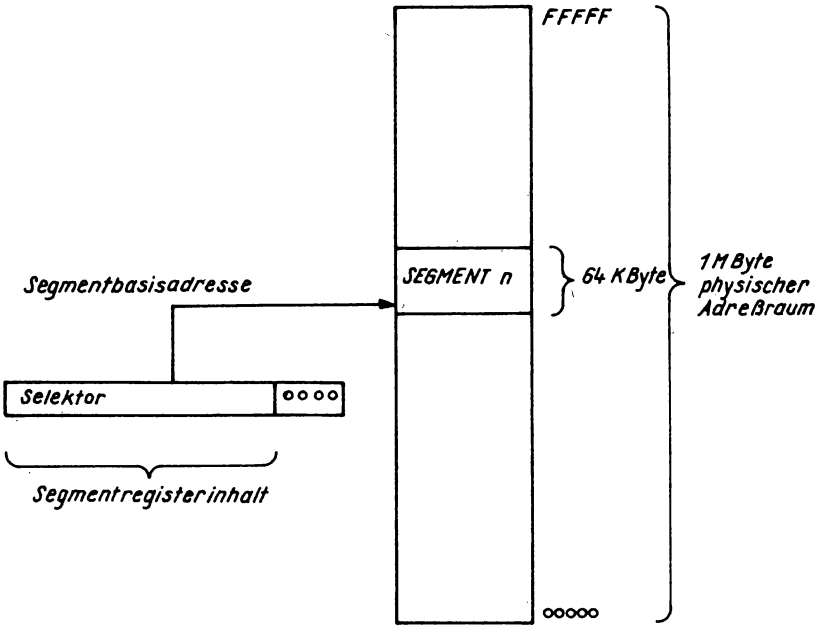


Bild 4.7. Segmentselektorinterpretation im RA-Mode des Mikroprozessors 80286

Die in einer Segmentdeskriptortabelle bzw. in einem Segmentdeskriptor-Cacheregister zusammengefaßten Informationen

Segmenttyp (Accessbyte),
Segmentbasisadresse und
Segmentlänge

werden zusammen mit den Adreßtranslationsmechanismen in den Abschnitten 5. (Adreßraum), 6. (Speicherverwaltung), 7. (Zugriffsschutz), 8. (Taskkonzept) und 9. (Interruptsystem) ausführlich behandelt.

4.3. Systemkonfigurationsregister (PVA-Mode)

Außer den bereits behandelten Segmentregistern und dem Segmentdeskriptor-Cacheregister existieren im Mikroprozessor 80286 die folgenden Systemkonfigurationsregister:

Globaldeskriptortabellenregister	GDTR,
Lokaldeskriptortabellenregister	LDTR,
Interruptdeskriptortabellenregister	IDRT und
Taskregister	TR.

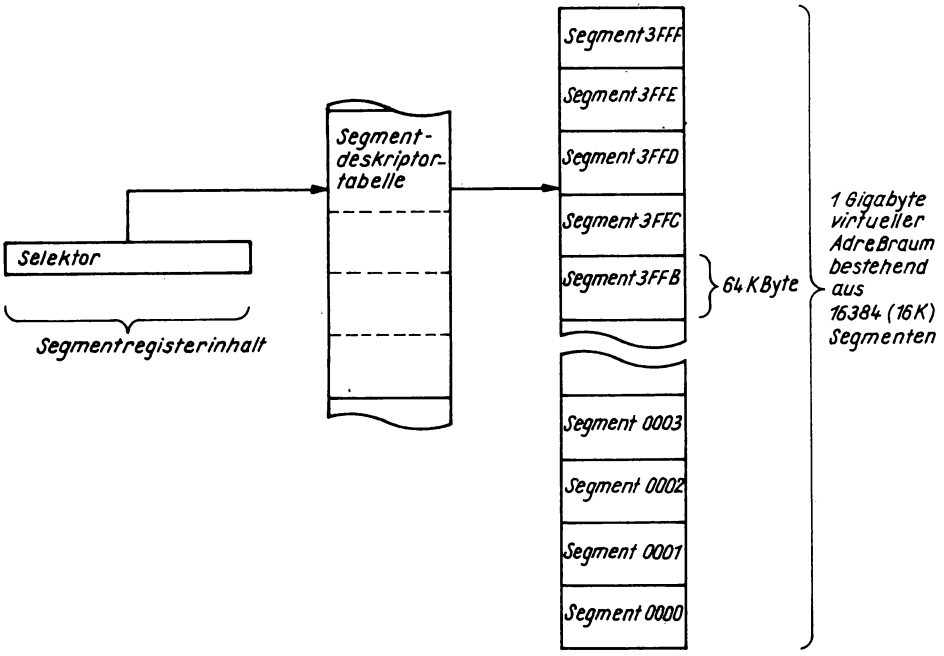


Bild 4.8. Segmentselektorinterpretation im PVA-Mode des Mikroprozessors 80286

Das Globaldeskriptortabellenregister (GDTR) spezifiziert direkt die physische Adresse und die Länge der Globaldeskriptortabelle (GDT) im Speicher des 80286-Mikrorechners.
Die Globaldeskriptortabelle enthält die Speichersegmentdeskriptoren der globalen Programm- und Datenspeichersegmente aller Tasks in einem konkreten Softwaresystem.
Das Lokaldeskriptortabellenregister (LDTR) spezifiziert über einen indirekten Verweis in der GDT die Anordnung der Lokaldeskriptortabelle (LDT) im Speicher des 80286-Mikrorechners.
Die Lokaldeskriptortabelle enthält die jeweils nur für eine Task gültigen Programm- und Datenspeichersegmentdeskriptoren.
Das Interrupttabellenregister (IDTR) spezifiziert direkt die physische Adresse und die Länge der Interruptdeskriptortabelle (IDT) im Speicher des 80286-Mikrorechners.
Die Interruptdeskriptortabelle zeichnet für die Zuordnung der externen und internen Interruptsignale zu den Interruptserviceroutinen verantwortlich.
Das Taskregister (TR) in der 80286-CPU schließlich zeigt immer auf einen TSS-Deskriptor. (TSS task state segment) in der Globaldeskriptortabelle (GDT), der der momentan aktiven Task zugeordnet ist.

Ein TSS-Deskriptor dient der Zuordnung von Taskstatusinformationen in einem 44 Byte langen Taskstatussegment zu einer Task.
Für jede Task in einem 80286-Softwaresystem existiert ein TSS-Deskriptor und ein zugehöriges Task-State-Segment.
Die etwas uneinheitliche Struktur zwischen GDTR und IDTR einerseits sowie LDTR und TR andererseits ergibt sich aus der Tatsache, daß normalerweise nur eine einzige Globaldeskriptortabelle und nur eine einzige Interruptdeskriptortabelle in einem konkreten Softwaresystem existieren, während die Anzahl der Lokaldeskriptortabellen und der Taskstatussegmente in der Regel mindestens so groß ist wie die Anzahl der lauffähigen Tasks.
Die Register LDTR und TR werden also bei jedem Taskwechsel neu geladen, während der Inhalt der Register GDTR und IDTR unverändert bleibt.
Für den LDTR- und TR-Cacheregisterteil gilt das im Abschnitt 4.2. (Segmentregister) Gesagte.
Die Veränderung des Inhalts der Systemkonfigurationsregister erfolgt mit speziell für diese Aufgabe vorgesehenen Befehlen (Abschnitt 13. Befehlssatz):

LGDT/SGDT	Laden/Rückspeichern GDTR	(load/store GDTR)
LIDT/SIDT	Laden/Rückspeichern IDTR	(load/store IDTR)
LLDT/SLDT	Laden/Rückspeichern LDTR	(load/store LDTR)
LTR /STR	Laden/Rückspeichern TR	(load/store TR)

Es muß an dieser Stelle angemerkt werden, daß die in die Register GDTR und IDTR vom Programmierer zu ladenden 24-Bit-Basisadressen die beiden einzigen direkt spezifizierten physischen Adressen in einem 80286-Softwaresystem sind, das im PVA-Mode arbeitet. Alle anderen Adressen im PVA-Mode werden über die Speicherverwaltungseinheit des Mikroprozessors berechnet (Abschn. 6. Speicherverwaltung).
Die Vorgabe der 24-Bit-Basisadressen für die beiden CPU-Register GDTR und IDTR muß immer vor der Umschaltung des 80286 vom RA-Mode in dem PVA-Mode erfolgen. Danach kommt die Umschaltung, und dann erfolgt erst die Vorgabe der Werte für die Register LDTR und TR (Abschn. 14. Systeminitialisierung).
Die behandelten Register GDTR, LDTR, IDTR und TR sind alle ausschließlich für die Nutzung im PVA-Mode vorgesehen, obwohl die eben angedeutete Initialisierungssequenz zeigt, daß die beiden Register GDTR und IDTR auch im RA-Mode zugänglich sind.
Die Systemkonfigurationsregister werden bezüglich ihres genauen Aufbaus und ihrer Funktion in den Abschnitten 5. (Adreßraum), 6. (Speicherverwaltung), 7. (Zugriffsschutz), 8. (Taskkonzept) und 9. (Interruptsystem) behandelt.

4.4. Flagregister

Das 16-Bit-Flagregister des Mikroprozessors 80286 beinhaltet jeweils sechs Status- und sechs Steuerflags. Vier Bitpositionen im Flagregister sind nicht belegt.

Flagregister des Mikroprozessors 80286:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

Die Statusflags des Mikroprozessors 80286 haben folgende Funktion:

CF	carry flag	Kennzeichnet einen Überlauf (Addition) oder ein Borgen (Subtraktion) bei 8- oder 16-Bit-Operanden nach einer arithmetischen Operation. Wird bei Rotations- und Verschiebeoperationen als Zwischenspeicher benutzt.
PF	parity flag	Kennzeichnet im Ergebnis einer Operation, ob die Anzahl der gesetzten Bits im resultierenden Binärwert gerade (PV=1, even) oder ungerade (PV=0, odd) ist.
AF	auxiliary carry flag	Kennzeichnet einen Überlauf (Addition) oder Borgen (Subtraktion) auf den unteren 4 Bit des Ergebnisses nach einer arithmetischen Operation mit 4-Bit-BCD-Zahlwerten.
ZF	zero flag	Signalisiert, ob im Ergebnis einer Operation der Wert 0 entstanden ist (ZF=1) oder nicht (ZF=0).
SF	sign flag	Signalisiert, ob im Ergebnis einer Operation ein negativer (SF=1) oder positiver (SF=0) Wert entstanden ist.
OF	overflow flag	Signalisiert, ob im Ergebnis einer Operation ein Vorzeichenüberlauf eingetreten ist (OF=1) oder nicht (OF=0).

Die Steuerflags des Mikroprozessors 80286 haben folgende Funktion:

IF	interrupt enable flag	Das Interruptsteuerflag (IF) dient der programmgesteuerten Aktivierung (IF=1) oder Deaktivierung (IF=0) des gesamten 80286-Interruptsystems (Abschn. 9. Interruptsystem).
TF	trap flag	Das Trapflag (TF) dient der programmgesteuerten Aktivierung (TF=1) oder Deaktivierung (TF=0) der im Mikroprozessor 80286 integrierten Betriebsart 'Einzelschrittabarbeitung' (single step mode). Diese Betriebsart ermöglicht die automatische Unterbrechung der Programmabarbeitung durch einen Softwareinterrupt (trap) nach jeweils einer kompletten Befehlsausführung.

Sie gestattet einem übergeordneten Trapbehandlungsprogramm die Organisation einer programmgesteuerten Fehlersuche mit Hilfe von Softwaredebuggern (Abschn. 9. Interruptsystem).

DF direction flag — Das Direktionsflag (DF) dient dem programmgesteuerten Wechsel der Vorwärts- oder Rückwärtsschrittabarbeitung bei Stringoperationen bezogen auf den Quell- und Zielindex (source index, destination index; SI oder DI bzw. SI und DI).
Wird der Wert von DF auf 1 gesetzt, erfolgt eine automatische Dekrementierung der Indexregister bei Zeichenkettenoperationen.
Hat das DF-Flagbit den Wert 0, erfolgt eine automatische Inkrementierung (Abschn. 13. Befehlssatz).

IOPL input/output privilege level flag — Die zwei IOPL-Flagbits gestatten die programmgesteuerte Auswahl einer von vier möglichen Ein-/Ausgabe-Privilegstufen im PVA-Mode des Mikroprozessors 80286.
Die IOPL-Vorrangstufen ermöglichen damit die Begrenzung von Ein-/Ausgabeoperationen auf bestimmte Privilegierungsstufen (Abschn. 10. Ein-/Ausgabe).

NT nested task flag — Das Taskverschachtelungsflag (NT) dient der programmgeteuerten Überwachung der Interruptverschachtelungsebenen beim Taskwechsel im PVA-Mode des Mikroprozessors 80286. Seine Funktion wird in den Abschnitten 8. (Taskkonzept) und 9. (Interruptsystem) genau behandelt.

Bei der Nutzung der Steuerflags des Mikroprozessors 80286 sind die folgenden Besonderheiten zu beachten:

- Das programmgesteuerte Setzen oder Rücksetzen der Steuerflags kann immer durch Modifikation des in den Stack abgespeicherten Flagregisterinhalts (push) im Speicher und anschließendes Rückspeichern (pop) des veränderten Flagworts in das Flagregister erfolgen.

- Bei jedem Setzen oder Rücksetzen einzelner oder mehrerer Bits im Flagregister sollte vom Programmierer darauf geachtet werden, daß die beim Mikroprozessor 80286 nicht benutzten und mit '0' oder '1' gekennzeichneten Flagbits aus Kompatibilitätsgründen zu künftigen Weiterentwicklungen dieses Mikroprozessors unverändert bleiben.

- Zum Setzen und Rücksetzen zweier Flags (IF und DF) existieren darüber hinaus die folgenden speziellen Befehle (Abschn. 13. Befehlssatz):

	Setzen (=1)	Rücksetzen (=0)
IF interrupt enable flag	STI	CLI
DF direction flag	STD	CLD

Wichtig ist, daß das Setzen und Rücksetzen des Interruptflags IF von der CPU-Steuerung des 80286 im PVA-Mode über die IOPL-Privilegierungsstufe überwacht wird, um nichtprivilegierten Anwenderprogrammen keine Veränderung des Interruptstatus zu ermöglichen.
Diese Überprüfung erfolgt sowohl beim Setzen und Rücksetzen des IF-Flags mit Hilfe der Befehle STI und CLI als auch beim Setzen und Rücksetzen des IF-Flags über eine Stackmanipulation (Abschn. 10. Ein-/Ausgabe).

4.5. Maschinenstatuswort

Das Maschinenstatuswort (MSW) beinhaltet beim Mikroprozessor 80286 in den unteren Bitpositionen vier unterschiedliche Steuer- und Statusindikatoren. Das Maschinenstatuswort kann über extra für diesen Zweck vorhandene 80286-Befehle gelesen und geschrieben werden (LMSW, SMSW).

Maschinenstatuswort des Mikroprozessors 80286:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	TS	EM	MP	PE

TS task switched flag
EM emulate flag
MP math present flag
PE protected mode enable/disable

Die Funktion des MSW-Bits 0 (protected mode enable/disable) wurde bereits im Abschnitt 3. (Betriebsarten) erläutert. Es dient der einmalig möglichen Umschaltung von der Betriebsart REAL ADDRESS MODE in die Betriebsart PROTECTED VIRTUAL ADDRESS MODE des Mikroprozessors 80286.

Die Bits 1, 2 und 3 (MP, EM, TS) des Maschinenstatusworts dienen der Steuerung der Zusammenarbeit einer 80286-CPU mit einem externen Numerikcoprozessor.

Im einzelnen übernehmen sie dabei die folgenden Aufgaben:

TS task switched flag

Das TS-Bit des Maschinenstatusworts in der 80286-CPU wird bei jedem Taskwechsel von der 80286-CPU automatisch gesetzt (TS=1).
Dadurch ist es möglich, von einem Programm aus über Softwaremaßnahmen zu erkennen, daß ein Taskwechsel stattgefunden hat, und zu entscheiden, ob der interne Zustand eines Numerikcoprozessors gerettet werden muß oder nicht.
Das TS-Bit dient damit der Verhinderung von Coprozessorfehlern, die ihre Ursache in der nicht abgeschlossenen Nutzung des Numerikcoprozessors in der vorhergehenden Task haben.
Außerdem existiert eine interne Zustandsüberwachung des TS-Flags, die sichert, daß immer, wenn ein Befehl für den externen Numerikcoprozessor von der 80286-CPU bei gesetztem TS-Bit (TS=1) dekodiert wird, eine automatische Generierung eines internen Softwareinterrupts (Interrupt 7) erfolgt. Dieser interne Interrupt gestattet es, in der Interruptbehandlungsroutine einen ggf. erforderlichen Statuswechsel - alte Task/neue Task - auch im Numerikcoprozessor abzuwickeln (Abschn. 9. Interruptsystem).
Das Rücksetzen des TS-Bits (=0) muß immer softwaregesteuert mit Hilfe des Befehls CLTS (clear task switched flag) erfolgen.

EM emulate flag

Das EM-Bit des Maschinenstatusworts zeigt an, ob eine für einen Numerikcoprozessor bestimmte Aufgabe softwaremäßig in der 80286-CPU emuliert werden soll (EM=1) oder nicht (EM=0).
Ist dieses Bit gesetzt (EM=1), führt ein Coprozessorbefehl zu einem internen Softwareinterrupt (Interrupt 7), der die den Coprozessorbefehl emulierende Behandlungsroutine einleitet.
Das EM-Bit wird ausschließlich über die Befehle LMSW und SMSW (load/store machine status word) gesetzt und rückgesetzt.
Das EM-Bit darf niemals gleichzeitig mit dem MP-Bit im Maschinenstatuswort gesetzt werden, das das physische Vorhandensein eines Numerikcoprozessors anzeigt (Numerikcoprozessor vorhanden: EM=0).

MP math present flag

Das gesetzte MP-Bit (MP=1) im Maschinenstatuswort der 80286-CPU zeigt das physische Vorhandensein eines externen Numerikcoprozessors an.
Auf den Numerikcoprozessor bezugnehmende Befehle werden dann automatisch an diesen externen Schaltkreis weitergeleitet.
Ist das MP-Bit rückgesetzt (MP=0), ist ein externer Numerikcoprozessor nicht vorhanden.

Für das Setzen und Rücksetzen des MP-Bits gilt das beim EM-Bit Gesagte.

Der Zustand der drei MSW-Flagbits TS, MP und EM ist nach dem Systemreset einheitlich 0.
Ist kein externer Numerikcoprozessor vorhanden, muß das EM-Bit nach Eintritt in den PVA-Mode gesetzt werden (EM=1), anderenfalls das MP-Bit (MP=1).
Bei jedem Setzen oder Rücksetzen einzelner oder mehrerer Bits im Maschinenstatuswort sollte vom Programmierer darauf geachtet werden, daß die beim Mikroprozessor 80286 nicht benutzten und mit '0' gekennzeichneten Bits aus Kompatibilitätsgründen zu künftigen Weiterentwicklungen dieses Mikroprozessors unverändert bleiben.
Auf die Funktion der MSW-Bits 1, 2 und 3 (MP, EM, TS) wird in den Abschitten 8. (Taskkonzept) und 9. (Interruptsystem) noch genauer eingegangen.

4.6. Befehlszähler

Der 16-Bit-Befehlszähler IP (instruction pointer) beinhaltet in beiden Betriebsarten des 80286 einen Adreßoffset, der den Versatz zwischen der Startadresse des aktuellen Programmkodesegments und dem nächsten auszuführenden Maschinenbefehl spezifiziert.

5. Adreßraum

Bei der Behandlung des 80286-Adreßraums muß zwischen dem physischen, logischen und virtuellen Adreßraum dieses Mikroprozessorsystems unterschieden werden:

physischer Adreßraum

Der physische Adreßraum ist der Adreßraum der physisch real zur Verfügung stehenden Speicheradressen.
Eine physische Adresse entspricht immer der realen Hardwareadresse in einem konkreten 80286-Mikroprozessorsystem.

logischer Adreßraum

Der logische Adreßraum ist der Adreßraum, der programmierlogisch aus Sicht des Programmierers (der Software) zur Verfügung steht.
Eine programmierlogische Adresse wird sowohl im RA-Mode als auch im PVA-Mode über Translationsmechanismen im Mikroprozessor in eine physische Adresse umgesetzt.

virtueller Adreßraum

Unter einem virtuellen Speichermodell wird die Implementation eines Adressierungsverfahrens verstanden, bei dem der programmierlogische Adreßraum wesentlich größer ist als der physische Adreßraum und bei dem automatisch - in Abhängigkeit von den Zugriffserfordernissen - Programm- und Datenstrukturen zwischen dem Hauptspeicher und dem Sekundärspeicher (in der Regel Hard-Disk) hin und her transportiert werden, unabhängig und unsichtbar für Anwenderprogramme.
Dem Programmierer erscheint es so, als würde der gesamte verfügbare Adreßraum nahezu unbegrenzt groß sein.
Eine virtuelle Adresse in einem Anwenderprogramm kann sich dann auf ein physisch im Moment noch nicht im Hauptspeicher befindlichen Programmkode- oder Datenteil beziehen.
Der Zugriff auf eine solche Adresse löst eine Aktion des Mikroprozessors aus - als Swapping bezeichnet -, die zum Auslagern eines nicht benötigten und zum Einlesen der adressierten, aber nicht im Hauptspeicher befindlichen Teile führt. Alle dazu notwendigen Prozeduren, einschließlich der Adreßzuordnung (virtuelle Adresse zu physischer Adresse), laufen vollautomatisch, von der Hardware des Mikroprozessors und von der Software des Betriebssystems gesteuert, ab.

Sowohl der physische als auch der logische bzw. virtuelle Adreßraum des Mikroprozessors 80286 besitzen - im Gegensatz zu einem linearen Adreßraum - eine segmentierte Struktur (Speichersegmente).
Sie soll den Entwurf, die Pflege und Wartung großer Softwarepakete durch den Zwang zur Modularisierung und Strukturierung vereinfachen.
Ein Segment ist eine logische Einheit eines nicht unterbrochenen Speicherbereichs. Es umfaßt beim Mikroprozessor 80286 einen 64 KByte großen Bereich, dessen Anfangsadresse durch den 16-Bit-Segmentselektor (segment selector) festgelegt ist. Innerhalb eines Segments wird eine Speicheradresse durch den 16-Bit-Adreßoffset (address offset) ausgewählt.

Bei der Bildung von Adressen im Mikroprozessorsystem 80286 wird grundsätzlich mit den beiden 16-Bit-Komponenten - Segmentselektor und Adreßoffset - gearbeitet.
Der Segmentselektor spezifiziert ein Segment im Sinne der eben beschriebenen Adreßraumsegmentierung, während der Adreßoffset innerhalb des ausgewählten, maximal 64 KByte großen Segments eine spezielle Byteadresse anwählt.

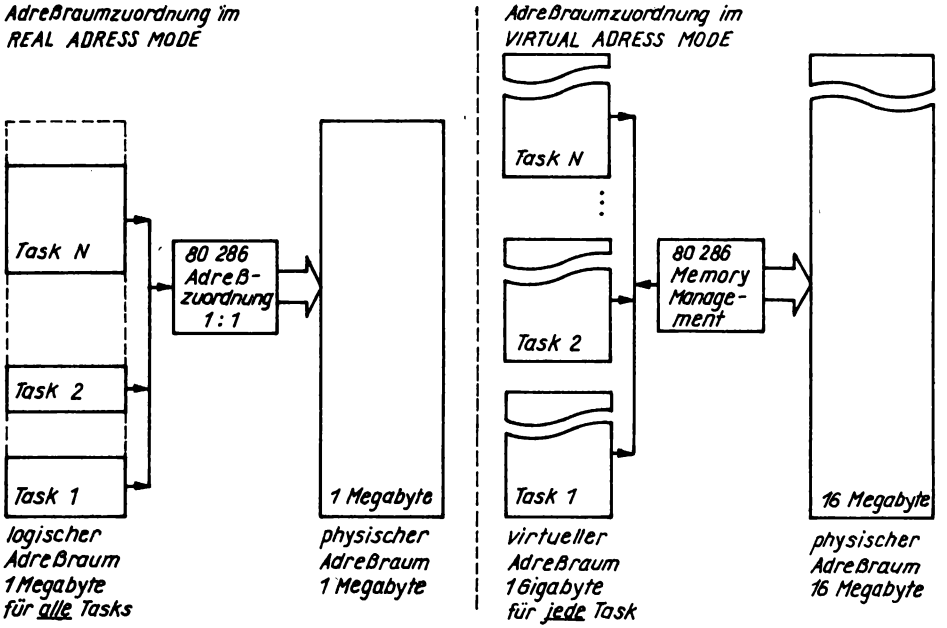


Bild 5.1. Adreßraumzuordnung im RA-Mode und PVA-Mode des Mikroprozessors 80286

Das Mikroprozessorsystem 80286 kennt, wie im Abschnitt 3. (Betriebsarten) beschrieben, zwei unterschiedliche Betriebsarten, den REAL ADDRESS MODE und den PROTECTED VIRTUAL ADDRESS MODE, die, wahlweise vom Programmierer einstellbar, stark voneinander differierende Adreßräume repräsentieren (Bild 5.1).
Für beide Betriebsarten gilt, daß der Adreßraum aus einer geschlossenen Folge einzeln adressierbarer Bytes besteht, die mit der Adresse 0 beginnen (Bild 5.2).
Bei der Adressierung von 16- oder 32-Byte-Worte stellt das niederwertigste Byte die jeweilige Wortadresse. Die höherwertigen Bytes folgen auf den nächsten Speicheradressen (Abschn. 11. Datenformate).

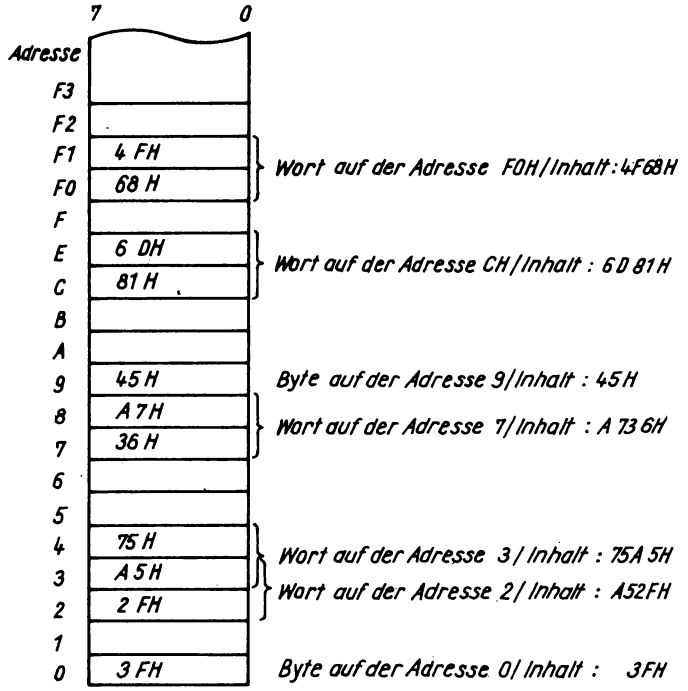


Bild 5.2. Byte- und Wortadressierung im Mikroprozessorsystem 80286

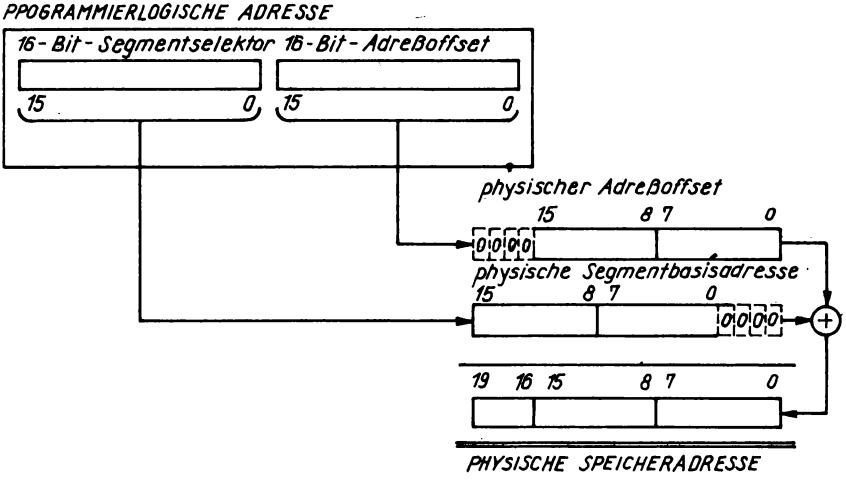


Bild 5.3. Adreßbildung im RA-Mode des Mikroprozessors 80286

Eine wichtige Rolle im Zusammenhang mit der Adreßraumbehandlung spielt, besonders im PVA-Mode, das beim Mikroprozessorsystem 80286 verwendete Taskkonzept (Abschn. 8. Taskkonzept).
Eine Task ist ein in Abarbeitung befindliches Programm mit allen die Programmabarbeitung kennzeichnenden Informationen. Dazu gehören der Programmkode, programmspezifische Datenstrukturen, Registerinhalte, Stackbelegungen, Statuszustände u.a.m.
Tasks werden verwendet, um die Arbeit des Mikroprozessors in einem Softwaresystem in möglichst unabhängige Teilkomponenten aufteilen zu können. Damit wird das Ziel verfolgt, ein aus dem Zusammenwirken von Teilkomponenten bestehendes Gesamtsystem leichter verstehen und behandeln zu können - vor allem auch in bezug auf den zeitlich intern rein sequentiellen, nach außen jedoch scheinbar parallelen Programmablauf.
Besondere Bedeutung kommt dem Taskkonzept in Multiuser-Time-Sharing-Systemen, in Multitasksystemen und in Echtzeitsystemen zu.

5.1. Adreßraum im RA-Mode

Die Betriebsart REAL ADDRESS MODE des Mikroprozessors 80286 ist dadurch gekennzeichnet, daß alle verwendeten Adressen reale physische Adressen darstellen.
Die Bildung der physischen Adressen erfolgt im RA-Mode nach dem im Bild 5.3 dargestellten Verfahren.
Ein 16-Bit-Adreßoffset wird zu der um vier Bit nach links verschobenen 16-Bit-Segmentbasisadresse addiert. Die entstehende 20-Bit-Adresse überstreicht einen physischen Adreßraum von insgesamt 1 MByte (00000H bis OFFFFFH ; H Hexadezimalkennzeichen).
Beide Komponenten der programmierlogischen Adresse, die bei der Bildung der physischen Speicheradressen im RA-Mode des 80286 Verwendung finden, entstehen unmittelbar bei der Befehlsabarbeitung.
Die erste Komponente, der 16-Bit-Adreßoffset, adressiert eine physische Speicheradresse in einem 64 KByte umfassenden Segment, dessen niedrigste Adresse 0000H und dessen höchste Adresse OFFFFH ist.
Die zweite Komponente der programmierlogischen Adresse im RA-Mode ist die 16-Bit-Segmentbasisadresse, deren Wert durch den Inhalt eines der vier Segmentregister des 80286-Registersatzes festgelegt wird.
Durch das im Bild 5.3 dargestellte Verfahren zur Berechnung der physischen Speicheradressen wird garantiert, daß die Segmentanfangspositionierung im 1-MByte-Adreßraum immer auf eine durch 16 teilbare physische Adresse fällt.
Durch gezielte Programmierung der Segmentbasisadresse ist eine Überlappung zweier oder mehrerer Segmente im RA-Mode denkbar (Bild 5.4).
Dabei sind allerdings ggf. Inkompatibilitätsprobleme der entstehenden Programm- und Datenstrukturen mit der für die Betriebsart PVA-Mode des 80286 vorgeschriebenen Adreßaufteilung zu beachten.
Zu beachten ist außerdem, daß in der Betriebsart RA-Mode keinerlei Adreß- oder Zugriffsberechtigungsüberprüfungen vom 80286 ausgeführt werden.

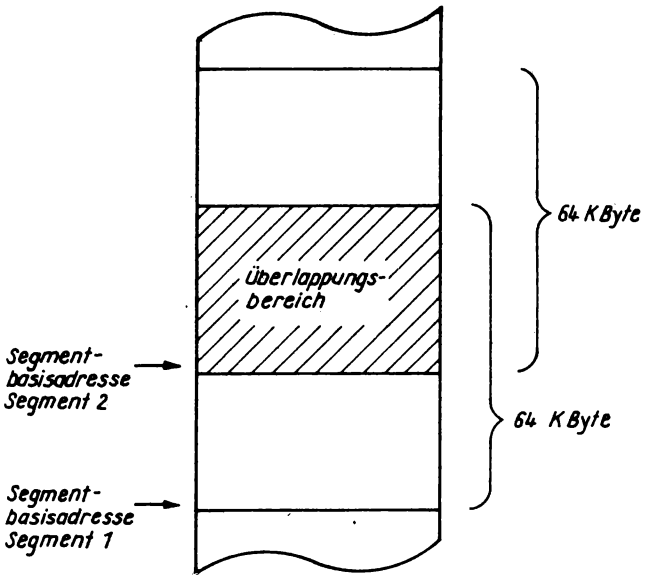


Bild 5.4. Segmentadreßüberlappung im RA-Mode des Mikroprozessors 80286

5.2. Adreßraum im PVA-Mode

Die Betriebsart PROTECTED VIRTUAL ADDRESS MODE des Mikroprozessors 80286 ist dadurch gekennzeichnet, daß alle verwendeten Adressen virtuellen Charakter haben. Der direkte Zugriff zu physischen Adressen wird nicht gestattet.
Der gesamte Speicheradreßraum des 80286 (Adreßraum des primären Hauptspeichers und der sekundären Massendatenspeicher) wird im PVA-Mode als ein großer, sogenannter virtueller Adreßraum betrachtet, der insgesamt 1 GByte (30-Bit-Adresse) pro Task umfaßt. Dieser virtuelle Adreßraum des 80286 wird auf einen 16 MByte (24-Bit-Adresse) großen physischen Adreßraum abgebildet, der über spezielle Translationsmechanismen (Speicherverwaltung) adressiert wird.
Die Bildung der virtuellen Adressen wird im PVA-Mode mit den gleichen Grundkomponenten wie im RA-Mode vorgenommen - einem 16-Bit-Segmentselektor und einem 16-Bit-Adreßoffset.
Der Unterschied zwischen den beiden Betriebsarten besteht in der unterschiedlichen Behandlung und Interpretation der Informationsinhalte dieser beiden Komponenten (Bilder 5.5 und 5.6).
Die aus den beiden 16-Bit-Komponenten entstehende virtuelle 32-Bit-Adresse wird im PVA-Mode des Mikroprozessors 80286 ähnlich interpretiert wie im RA-Mode.

Der 16-Bit-Segmentselektor (segment selector) wird meistens aus einem Segmentregister entnommen. Er dient der Auswahl eines Speichersegments. Der 16-Bit-Adreßoffset (address offset) dient der Auswahl einer einzelnen Speicherposition innerhalb des spezifizierten Segments.
Da aus dem 16-Bit-Segmentselektor zwei Bits (RPL) für Zugriffsschutzzwecke herausgelöst werden, steht effektiv eine virtuelle 30-Bit-Adresse zur Verfügung, die den virtuellen 1-GByte-Adreßraum des Mikroprozessors 80286 im PVA-Mode überstreicht.
Der unterschiedliche Informationsinhalt der beiden Adreßkomponenten kommt im PVA-Mode durch die vom RA-Mode stark abweichende Nutzung des 16-Bit-Segmentselektors zum Ausdruck. Sie ist, wie im Bild 5.7 dargestellt, aus den drei Teilkomponenten RPL-Feld, TI-Bit und INDEX-Feld aufgebaut.
Sie dienen zur Vorgabe von vier Privilegierungsstufen (RPL-Feld), zur Auswahl eines globalen oder lokalen Adreßraums (TI-Bit) und zur Ermittlung der Segmentlokalisierung über eine Deskriptortabelle mit den physischen Segmentanfangsadressen (INDEX-Feld).

RPL-Feld

Die beiden RPL-Bits (requested privilege level) dienen der Festlegung einer Aufrufprivilegierungsstufe, die einem Speicherzugriff bei der Programmabarbeitung im Mikroprozessor 80286 zugeordnet werden kann. Die Zuordnung wird vom Programmierer bei der Systemstrukturierung konzipiert.
Sie kann eine der folgenden vier Privilegierungsstufen umfassen (Beispiel):

Level 0 Betriebssystemkern,
Level 1 Betriebssystemserviceroutinen,
Level 2 Applikationsprogrammserviceroutinen,
Level 3 Applikationsprogramme.

Die beiden RPL-Bits werden bei der Adreßbildung aus dem 16-Bit-Segmentselektor herausgelöst und ausschließlich für die genannten Speicherschutzfunktionen verwendet (Abschnitte 6. Speicherverwaltung und 7. Zugriffsschutz).

TI-Bit

Das TI-Bit (table indicator) unterteilt den virtuellen Adreßraum einer Task im PVA-Mode in zwei unterschiedliche Teile - einen globalen (TI=0) und einen lokalen (TI=1).
Der globale Teil des Adreßraums einer Task wird für globale Programme und Daten genutzt, die einen taskübergreifenden Charakter besitzen.
Im lokalen Teil des Adreßraums werden immer der auf eine Task bezogene lokale Programmkode und die lokalen Daten angeordnet.
Jeder der beiden Teile des Adreßraums hat, resultierend aus dem folgenden 13-Bit-INDEX-Feld, insgesamt 8192 (8 K) einzeln auswählbare Segmente.
Werden lokaler und globaler Adreßraum zusammengenommen, existieren 16384 (16 K) unterschiedliche Segmente einer Task im PVA-Mode des 80286.

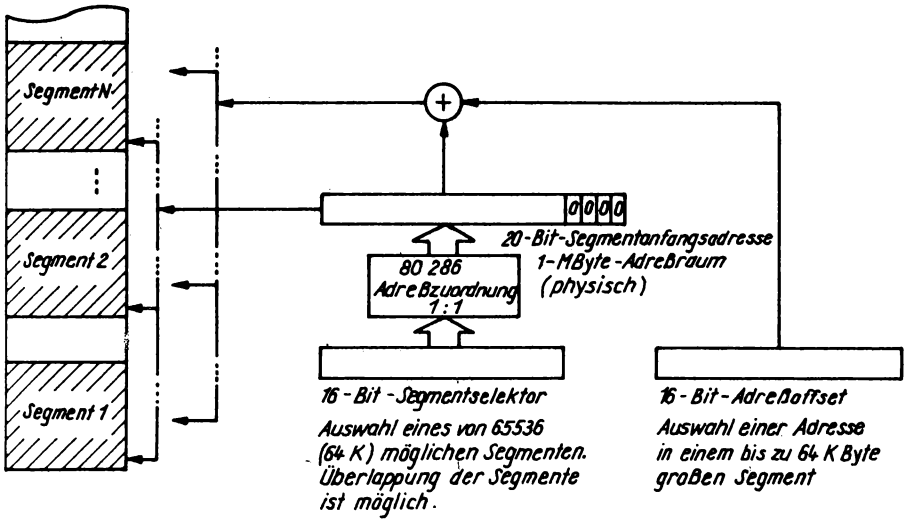


Bild 5.5. Segmentselektor- und Adreßoffsetinterpretation im RA-Mode des Mikroprozessors 80286

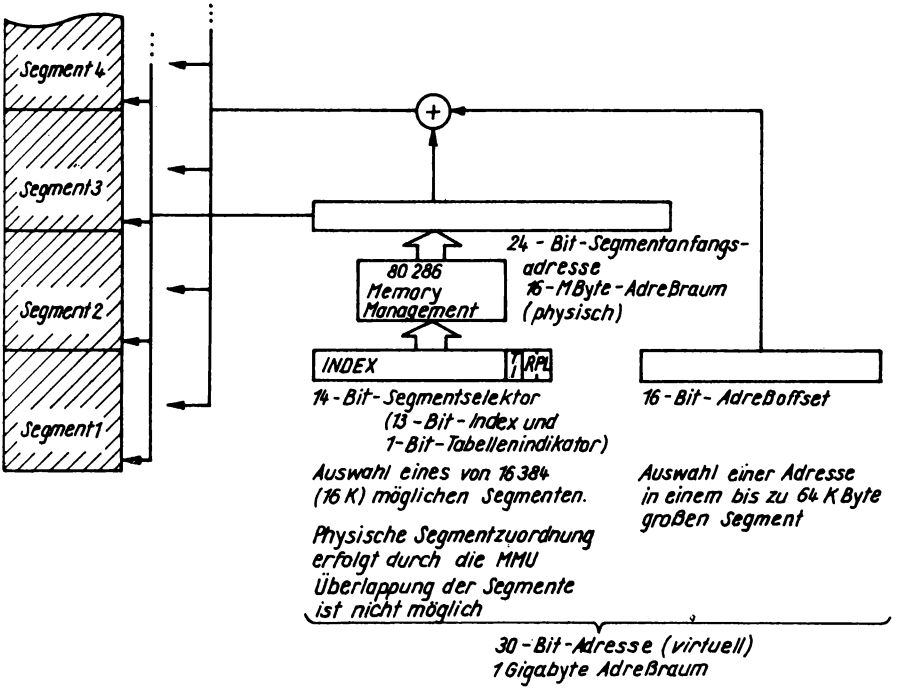


Bild 5.6. Segmentselektor- und Adreßoffsetinterpretation im PVA-Mode des Mikroprozessors 80286

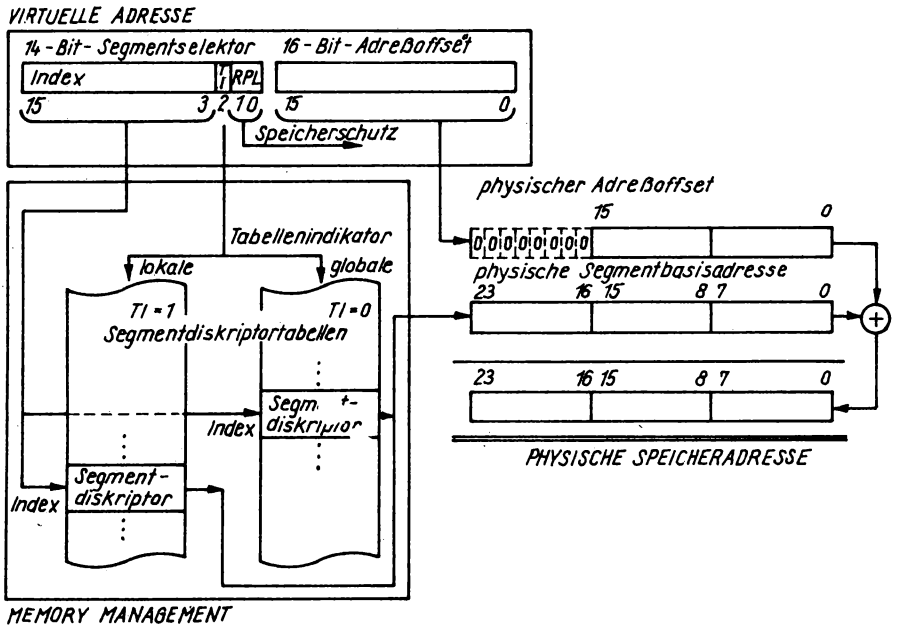


Bild 5.7. Adreßbildung im PVA-Mode des Mikroprozessors 80286

INDEX-Feld Über das 13-Bit-INDEX-Feld des 16-Bit-Segmentselektors erfolgt die physische Segmentlokalisierung.
Es dient als Pointer in zwei die physischen Segmentanfangsadressen enthaltenden Deskriptortabellen, einer globalen (TI=0) und einer lokalen (TI=1).
Die Deskriptortabellen sind verantwortlich für die jeweils aktuelle Zuordnung von virtuellen Adressen zu physischen Adressen im Speicherverwaltungssystem des Mikroprozessors 80286.
Diese Zuordnung, auch als 'mapping' oder 'memory management' bezeichnet, ist durch den programmgesteuerten Zugriff auf die Deskriptortabellen von entsprechenden Serviceprogrammen notwendigenfalls dynamisch veränderbar (Abschn. 6. Speicherverwaltung).

6. Speicherverwaltung

Die Grundlage des 80286-Speicherverwaltungssystems bildet die im Abschnitt 5. (Adreßraum) bereits ausführlich behandelte Speichersegmentierung. Sie geht davon aus, daß der Speicherraum des Mikroprozessors 80286 keine linear durchgehende Struktur besitzt, sondern aus einer definierten Anzahl von jeweils bis zu 64 KByte großen Speichersegmenten besteht.
Die Auswahl eines Speichersegments wird in einem 80286-Mikroprozessorsystem entkoppelt von der Auswahl einer Adresse innerhalb eines angesprochenen Segments. Dadurch werden alle Programme und Datenstrukturen in beiden 80286-Betriebsarten automatisch verschiebbar (relocatable), bezogen auf die Anordnung in einem bestimmten Segment.
Das genaue Bildungsverfahren der physischen Adresse und die Behandlung bzw. Verwaltung der Speichersegmente unterscheidet sich allerdings in beiden möglichen Betriebsarten des Mikroprozessors 80286 erheblich voneinander.

6.1. Speicheradressierung im RA-Mode

In der Betriebsart REAL ADDRESS MODE des Mikroprozessors 80286 erfolgt die Speicheradressierung nach einem sehr einfachen Verfahren (Abschn. 5.1. Adreßraum RA-Mode).
An den aus dem verwendeten Segmentregister entnommenen 16-Bit-Segmentselektor (segment selector) werden intern vier binäre Nullen rechts angefügt, so daß eine physische 20-Bit-Segmentanfangsadresse innerhalb eines 1 MByte großen Adreßraums entsteht.
Zu der physischen 20-Bit-Segmentanfangsadresse wird von der 80286-CPU bei einem Speicherzugriff automatisch der 16-Bit-Adreßoffset rechtsbündig addiert, um die endgültige physische 20-Bit-Adresse einer adressierten Speicherposition innerhalb des Segments zu erhalten (Bild 5.3).
Von einem echten Speicherverwaltungssystem mit Zugriffsschutzfunktionen und unterschiedlichen Privilegierungsstufen kann im RA-Mode keine Rede sein.

6.2. Speicherverwaltung im PVA-Mode

Im PROTECTED VIRTUAL ADDRESS MODE des Mikroprozessors 80286 existiert ein hochorganisiertes Speicherverwaltungssystem (memory management) mit allen Leistungscharakteristiken zur Zugriffsschutzüberprüfung unter den Bedingungen unterschiedlicher Privilegierungsstufen.
Das Speicherverwaltungssystem des Mikroprozessorsystems 80286 wurde im Gegensatz zu Speicherverwaltungssystemen anderer Mikroprozessorfamilien komplett in den CPU-Baustein integriert. Es befindet sich also nicht in einem externen MMU-Schaltkreis (MMU memory management unit), sondern in der 80286-CPU selbst.
Grundsätzlich gilt, daß das Speicherverwaltungssystem des Mikroprozessors 80286 die konkrete Zuordnung einer programmierlogischen Adresse aus dem virtuellen Adreßraum zu einer realen physischen Speicheradresse übernimmt. Dadurch wird der programmierlogische Adreßraum unabhängig (dynamically relocatable) vom physischen Speicheradreßraum in einem konkreten 80286-Mikroprozessorsystem.

Der Zuordnungsprozeß selbst wird von der Systemsoftware, also dem Betriebssystem der 80286-CPU, beherrschbar. Das erlaubt, die Programmerarbeitung sowohl für RAM-orientierte große Betriebssysteme mit veränderlichen Strukturen als auch für EPROM-residente kleine statische Echtzeitstrukturen weitgehend unabhängig von konkreten Speicheradressen vorzunehmen (RAM random access memory; EPROM electrically programmable read only memory).
Ausgangspunkt der Speicherverwaltung des Mikroprozessors 80286 in der Betriebsart PVA-Mode sind folgende Überlegungen:

- Der virtuelle Adreßraum des Programmierers soll bei Bedarf wesentlich größer sein können als der in einem 80286-Mikroprozessorsystem verfügbare physisch real adressierbare Hauptspeicherbereich.

- Die Abbildung des virtuellen Adreßraums auf den physisch real vorhandenen Arbeitsspeicher wird vom Speicherverwaltungssystem übernommen.

- Da der von einem Programmsystem verwendete virtuelle Adreßraum größer sein kann als der physisch real verfügbare Arbeitsspeicher, müssen ggf. Teile der Programm- und Datenstrukturen auf einem Hintergrundspeicher ausgelagert werden (swapping).

- Bezieht sich ein Zugriff eines in Ausführung begriffenen Programms auf Teile, die sich nicht im Arbeitsspeicher befinden, muß der gerade ausgeführte Befehl abgebrochen, das angesprochene Programm- oder Datensegment vom Betriebssystem in den Arbeitsspeicher geholt, die Segmentierung neu errichtet und der vorher abgebrochene Befehl nochmals ausgeführt werden.

Neben der reinen Adreßzuordnung muß also, wenn notwendig, auch die Ein-/Auslagerung von Programmen und Datenstrukturen durch das Speicherverwaltungssystem unterstützt werden.
Durch die virtuellen Adressierungsmechanismen kann die Softwareerarbeitung weitgehend entkoppelt von den speziellen Eigenschaften eines konkreten Mikrocomputersystems erfolgen.
Neben der Adreßzuordnung (mapping) und der Ein-/Auslagerung (swapping) spielen im Zusammenhang mit dem Speicherverwaltungssystem Fragen des Zugriffsschutzes (protection) und des Taskkonzepts eine herausragende Rolle. Wegen der besonderen Bedeutung der letztgenannten beiden Themenkomplexe wurde ihnen jeweils ein eigenständiger Abschnitt zugedacht (Abschnitte 7. Zugriffsschutz und 8. Taskkonzept).

6.2.1. Speichersegmentdeskriptoren

Die Grundlage für das gesamte Speicherverwaltungssystem des Mikroprozessors 80286 im PVA-Mode bildet die Zuordnung eines beschreibenden Speichersegmentdeskriptors zu jedem verwendeten Speichersegment (Bild 6.1).

In Verbindung mit der bereits im Abschnitt 5.2. (Adreßraum PVA-Mode) behandelten Aufteilung des Segmentselektors in die Teilkomponeneten:

INDEX-Feld (13 Bit) Auswahl eines Speichersegmentdeskriptors innerhalb einer von zwei möglichen Deskriptortabellen - einer globalen und einer lokalen,

TI-Feld (1 Bit) Auswahl einer von zwei möglichen Deskriptortabellen - einer globalen und einer lokalen und

RPL-Feld (2 Bit) Festlegung einer Aufrufprivilegierungsstufe (requested privilege level),

dient ein Speichersegmentdeskriptor zur Definition aller für das Speicherverwaltungssystem (mapping, swapping), den Zugriffsschutz (protection) und das Taskkonzept notwendigen Informationen.

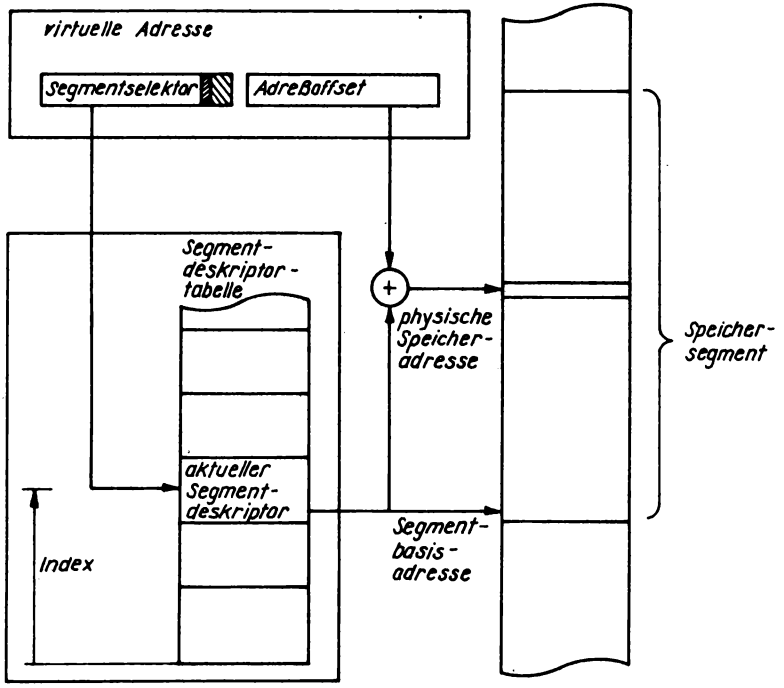
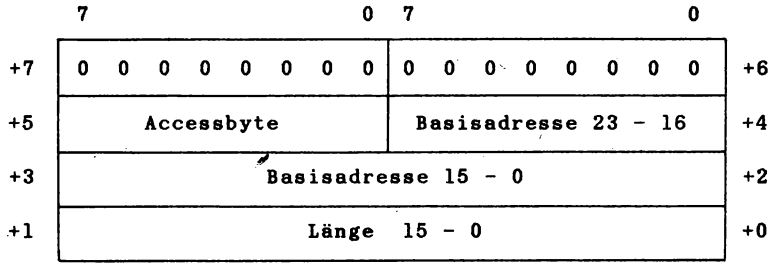


Bild 6.1. Auswahl eines Speichersegmentdeskriptors bei der Adreßbildung im PVA-Mode des Mikroprozessors 80286

Der ein Speichersegment beschreibende Speichersegmentdeskriptor enthält in insgesamt acht informationstragenden Bytes folgende Parameter:



Länge — Die Länge eines Speichersegments wird als 16-Bit-Wert angegeben.
Speichersegmente können in beliebiger Abstufung zwischen 1 Byte und 64 KByte lang sein. Sie können also den realen Erfordernissen, bezüglich der notwendigen Speichersegmentlänge, exakt angepaßt werden. Das sichert eine sehr effektive Ausnutzung des verfügbaren Speichers und die sofortige Erkennung von Speicherbereichsüberschreitungen durch fehlerhafte Programme.

Basisadresse — Die 24-Bit-Basisadresse ist die physische Anfangsadresse eines Speichersegments in einem 16 MByte großen physischen Adreßraum.
Speichersegmente können auf einer beliebigen physischen Adresse beginnen.

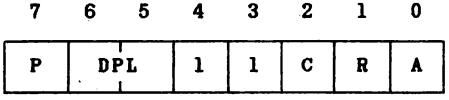
Accessbyte — Das Accessbyte (Zugriffsbyte) eines Speichersegments enthält wichtige Informationen für die Zugriffsschutzsicherung, die in der Folge genau zu erläutern sind.

0-Bytes — Die höchstwertigen beiden Bytes jedes Deskriptors werden beim Mikroprozessor 80286 nicht genutzt. Sie sollten beide immer den Wert 0 haben, um die Softwarekompatibilität zu weiterentwickelten CPU-Typen dieser Mikroprozessorfamilie abzusichern.

Das Accessbyte eines Speichersegmentdeskriptors beschreibt den Typ (Programmkode, Daten, Stack usw.) und die Art des erlaubten Zugriffs (Lesen, Schreiben, Ausführen usw.) auf ein Speichersegment.
Im Accessbyte des Speichersegmentdeskriptors wird ferner die Deskriptorprivilegierungsstufe DPL (descriptor privilege level) des Speichersegments abgelegt.
Die Deskriptorprivilegierungsstufe spielt eine wichtige Rolle beim Zugriffsschutz. Sie kennzeichnet die Privilegierungsstufe des Speichersegments, das durch den vorliegenden Deskriptor beschrieben wird.

Das Accessbyte enthält außerdem Informationsbits, die kennzeichnen, ob sich das Speichersegment im Moment im Arbeitsspeicher befindet oder ausgelagert wurde (P present) und ob auf das Speichersegment zugegriffen wurde oder nicht (A accessed). Die beiden Informationsbits P und A unterstützen insbesondere das bei virtuellen Adressierungsverfahren ggf. notwendige Ein-/Auslagern von Programm- und Datenstrukturen (swapping).
Zwischen dem Speichersegmentdeskriptor eines Programmkodesegments und dem Speichersegmentdeskriptor eines Daten- bzw.Stacksegments existiert in zwei informationstragenden Bits ein Unterschied, der den Tafeln 6.1 und 6.2 entnommen werden kann. Die Tafel 6.3 schließlich gibt eine Übersicht über alle möglichen hexadezimal kodierten Werte im Accessbyte eines Speichersegmentdeskriptors.

Tafel 6.1. Übersicht über den Accessbyteaufbau eines Speichersegmentdeskriptors für ein Programmkodesegment



P Präsenzbit (present)

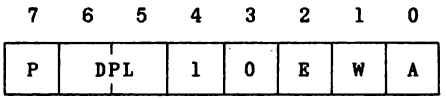
P=0	Das Programmkodespeichersegment befindet sich nicht im Hauptspeicher.
P=1	Das Programmkodespeichersegment befindet sich im Hauptspeicher.
Bemerkung:	Wenn ein nicht präsentes Speichersegment angesprochen wird, erfolgt über ein internes Interruptsignal (Interrupt 11) eine Programmunterbrechungsmeldung, die es gestattet, über Softwaremaßnahmen ein nicht im Primärspeicher befindliches Programmkodesegment in den Hauptspeicher zu laden und danach den zur Programmunterbrechung führenden Befehl noch einmal auszuführen (Abschn. 9. Interruptsystem). Das Setzen und Rücksetzen des P-Bits im Accessbyte muß vom Programmierer übernommen werden, es erfolgt nicht automatisch.

DPL Deskriptorprivilegierungsstufe (descriptor privilege level)

DPL=00	Privilegierungsstufe 0
DPL=01	Privilegierungsstufe 1
DPL=10	Privilegierungsstufe 2
DPL=11	Privilegierungsstufe 3

Bemerkung:	Die Deskriptorprivilegierungsstufe gibt die Privilegierungsstufe des durch den Speichersegmentdeskriptor beschriebenen Programmkodesegments an. Es handelt sich hier also um die Privilegierungsstufe des Speichersegments und nicht um die des Deskriptors.
C Anpassungsbit (conforming)	
C=0	Das Programmkodesegment wird beim Aufruf mit einer festen, durch den DPL-Wert spezifizierten Privilegierungsstufe ausgeführt.
C=1	Das Programmkodesegment wird beim Aufruf mit der Privilegierungsstufe abgearbeitet, die die aufrufende Prozedur besitzt. Es paßt sich an unterschiedliche Privilegierunsstufen an (es geht konform). Conforming-Programmkodesegmente haben ihre besondere Bedeutung u.a. bei der Interruptbehandlung (Abschn. 9. Interruptsystem).
R Lesebit (readable)	
R=0	Auf das Programmkodesegment ist der Lesezugriff nicht erlaubt, der Inhalt des Segments darf nur ausgeführt werden (execute only).
R=1	Auf das Programmkodesegment ist der Lesezugriff erlaubt (execute and read).
A Zugriffsbit (accessed)	
A=0	Auf den Speichersegmentdeskriptor bzw. das Speichersegment wurde nicht zugegriffen (read/write/executed).
A=1	Auf den Speichersegmentdeskriptor bzw. das Speichersegment wurde zugegriffen (read/write/executed).
Bemerkung:	Wenn das Zugriffsbit einmal von der 80286-CPU automatisch gesetzt wurde (A=1), wird es vom Mikroprozessor nicht mehr auf 0 zurückgesetzt. Das ist Aufgabe des Programmierers, der das Zugriffsbit in einer Behandlungsroutine auswerten möchte (z.B. in einer zeitzyklischen Swappingroutine).

Tafel 6.2. Übersicht über den Accessbyteaufbau eines Speichersegmentdeskriptors für ein Datenspeichersegment



P Präsenzbit (present)

P=0, P=1 siehe Programmkode-Speichersegmentdeskriptor

DPL Deskriptorprivilegierungsstufe (descriptor privilege level)

DPL=00	Privilegierungsstufe 0
DPL=01	Privilegierungsstufe 1
DPL=10	Privilegierungsstufe 2
DPL=11	Privilegierungsstufe 3

Bemerkung: siehe Programmkode-Speichersegmentdeskriptor

E Ausdehnungsrichtungsbit (expansion direction)

E=0 Das Datensegment wächst nach oben, es belegt den Adreßbereich:

(Basisadresse) bis (Basisadresse + Länge)

E=1 Das Datensegment wächst von oben nach unten, es belegt den Adreßbereich:

(Basisadresse + Länge + 1) bis (Basisadresse + OFFFFH)

Bemerkung: Das Ausdehnungsrichtungsbit zeigt die Wachstumsrichtung des Daten-, Extradaten- oder Stackspeichersegments an. Stackdatensegmente wachsen im allgemeinen in Richtung fallender Adressen, während Daten- und Extradatensegmente im allgemeinen sich in Richtung steigender Adressen ausdehnen.

W Schreibbit (writable)

W=0	Das Speichersegment darf nur gelesen werden - read only.
W=1	Das Speichersegment darf gelesen und geschrieben werden.

A Zugriffsbit (accessed) A=0, A=1 siehe Programmkode-Speichersegmentdeskriptor

Tafel 6.3. Übersicht über die Kodierung von Speichersegmentdeskriptoren (hexadezimal kodiert)

P=0				P=1				Bemerkung
DPL=				DPL=				
0	1	2	3	0	1	2	3	
10	30	50	70	90	B0	D0	F0	Expand-up, read only, ignored, Daten
11	31	51	71	91	B1	D1	F1	Expand-up, read only, accessed, Daten
12	32	52	72	92	B2	D2	F2	Expand-up, writable, ignored, Daten
13	33	53	73	93	B3	D3	F3	Expand-up, writable, accessed, Daten
14	34	54	74	94	B4	D4	F4	Expand-down, read only, ignored, Daten
15	35	55	75	95	B5	D5	F5	Expand-down, read only, accessed, Daten
16	36	56	76	96	B6	D6	F6	Expand-down, writable, ignored, Daten
17	37	57	77	97	B7	D7	F7	Expand-down, writable, accessed, Daten
18	38	58	78	98	B8	D8	F8	Non-conform, no read, ignored, Programm
19	39	59	79	99	B9	D9	F9	Non-conform, no read, accessed, Programm
1A	3A	5A	7A	9A	BA	DA	FA	Non-conform, readable, ignored, Programm
1B	3B	5B	7B	9B	BB	DB	FB	Non-conform, readable, accessed, Programm
1C	3C	5C	7C	9C	BC	DC	FC	Conforming, no read, ignored, Programm
1D	3D	5D	7D	9D	BD	DD	FD	Conforming, no read, accessed, Programm
1E	3E	5E	7E	9E	BE	DE	FE	Conforming, readable, ignored, Programm
1F	3F	5F	7F	9F	BF	DF	FF	Conforming, readable, accessed, Programm

6.2.2. Systemdeskriptoren

Neben den ein allgemeines Speichersegment beschreibenden Speichersegmentdeskriptoren (meist werden sie abgekürzt einfach als 'Segmentdeskriptoren' bezeichnet) existieren beim Mikroprozessor 80286 außerdem besonderen Aufgabenklassen (Adreßraumbehandlung, Interruptarbeit, Taskkonzept, Zugriffsschutz) zugeordnete Systemdeskriptoren.
Sie unterscheiden sich weniger in ihrem Aufbau oder ihrer Struktur als vielmehr in der Belegung und Interpretation der verschiedenen informationstragenden Einheiten (Bild 6.2).
Es existieren folgende sechs Klassen von Systemdeskriptoren:

TSS-Deskriptor — Beschreibt ein systemspezifisches TSS-Speichersegment mit Taskstatusinformationen (TSS task state segment).

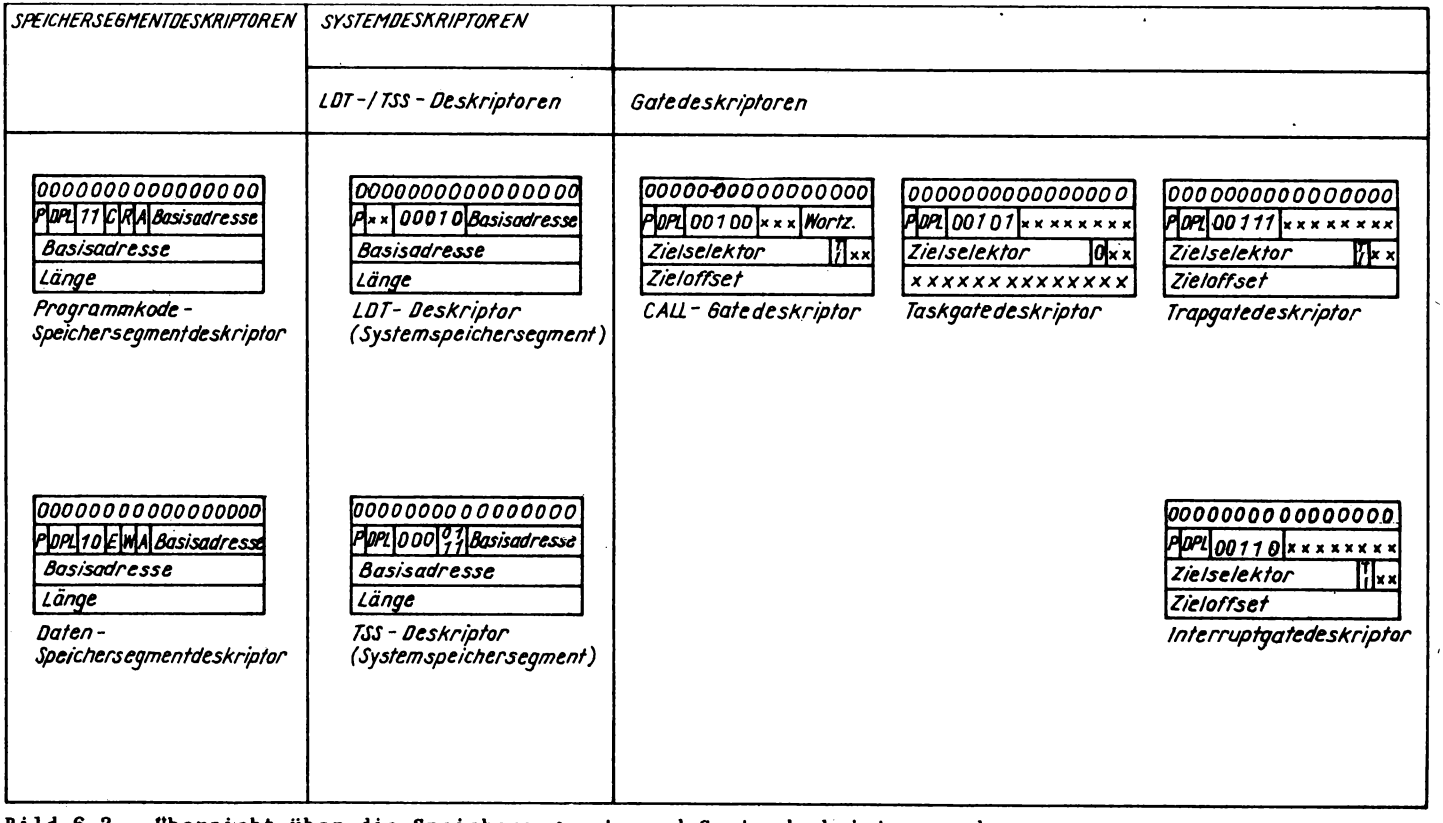


Bild 6.2. Übersicht über die Speichersegment- und Systemdeskriptoren des Mikroprozessors 80286

LDT-Deskriptor	Beschreibt ein systemspezifisches LDT-Speichersegment mit der Lokaldeskriptortabelle.
CALL-Gatedeskriptor	Steuerdatenstruktur für den von der 80286-CPU überwachten Übergang von einer Prozedur mit einer fest definierten Privilegierungsstufe auf eine Prozedur mit einer höheren oder gleichen Privilegierungsstufe innerhalb ein und derselben Task.
Taskgatedeskriptor	Steuerdatenstruktur für die von der 80286-CPU überwachte Abwicklung des Taskwechsels.
Interruptgatedeskriptor	Steuerdatenstruktur zur Sicherung der von der 80286-CPU überwachten Interruptbehandlung von externen Interruptsignalen ohne Taskwechsel.
Trapgatedeskriptor	Steuerdatenstruktur zur Sicherung der von der 80286-CPU überwachten Interruptbehandlung von internen Interruptsignalen ohne Taskwechsel.

Der TSS-Deskriptor dient der Lokalisierung des eine Task beschreibenden TSS-Speichersegments. Das 44 Byte lange Task-State-Segment (TSS) enthält solche Informationen wie die Basisadresse der zu einer Task gehörenden Lokaldeskriptortabelle, den aktuellen Taskstatus (Registerzustände), Stackpointerwerte u.a.m.
Im TSS-Deskriptor selbst befinden sich dabei zwei Kennzeichenbits, die anzeigen, ob die jeweilige Task im Moment aktiv ist oder nicht.
TSS-Deskriptoren können sich ausschließlich in der Globaldeskriptortabelle (GDT) befinden.
Das TSS-Konzept gestattet den außerordentlich schnellen - von der Hardware des 80286 voll unterstützten - Taskwechsel. Das Taskkonzept des Mikroprozessors 80286 wird ausführlich im Abschnitt 8. (Taskkonzept) behandelt.
Ein LDT-Deskriptor spezifiziert ein Datenspeichersegment, die sogenannte Lokaldeskriptortabelle (LDT), die alle Speichersegment- und Systemdeskriptoren einer Task aufnimmt, die sich auf Programm- und Datenspeichersegmente beziehen und die ausschließlich lokalen, also auf eine einzelne Task bezogenen Charakter haben.
LDT-Deskriptoren befinden sich immer in der Globaldeskriptortabelle (GDT). Sie werden in den Abschnitten 6.2.3. (Deskriptortabellen) und 8. (Taskkonzept) näher behandelt.
Gatedeskriptoren beschreiben kein Speichersegment, sondern stellen eine eigenständige Steuerdatenstruktur dar.
Gatedeskriptoren (CALL-Gate-, Taskgate-, Interruptgate- und Trapgatedeskriptoren) werden verwendet, um

- einer Prozedur einer Task zu ermöglichen, andere Prozeduren der gleichen Task mit höherer oder gleicher Privilegierungsstufe aufzurufen und dabei zu überwachen, ob die aufrufende Prozedur das Recht hat, die aufgerufene Prozedur zu verwenden (CALL-Gatedeskriptor),

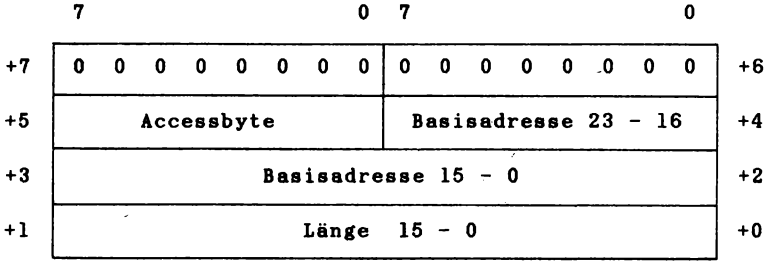
- sicherzustellen, daß bestimmte Prozeduren einer Task nur über die Vermittlung eben eines Gatedeskriptors ein anderes Programm aufrufen können (CALL-Gatedeskriptor),

- indirekte Zieladressen in einem Gatedeskriptor ändern zu können, ohne die Adresse in der aufrufenden Prozedur innerhalb einer Task verändern zu müssen (CALL-Gatedeskriptor),

- einen Taskwechsel einzuleiten (Taskgatedeskriptor) und

- die Interruptbearbeitung bei externen oder internen Interruptsignalen über entsprechende Serviceroutinen mit oder ohne Taskwechsel abzuwickeln (Task-, Interrrupt- und Trapgatedeskriptor).

Gatedeskriptoren werden in den Abschnitten 7. (Zugriffsschutz), 8. (Taskkonzept) und 9. (Interruptsystem) genauer behandelt.
TSS- und LDT-Systemdeskriptoren enthalten in insgesamt acht Bytes folgende Parameter:



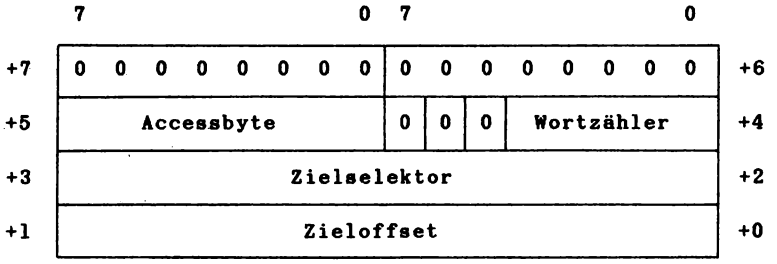
Länge Die Länge eines TSS- oder LDT-Systemdeskriptors wird als 16-Bit-Wert angegeben. Er wird, wie bei Speichersegmentdeskriptoren, zur Vorgabe einer maximalen Länge des TSS- bzw. LDT-Speichersegments verwendet.

Basisadresse Die 24-Bit-Basisadresse ist bei TSS- und LDT-Deskriptoren die physische Anfangsadresse des jeweiligen Systemspeichersegments in einem 16 MByte großen physischen Adreßraum.

Accessbyte Das Accessbyte (Zugriffsbyte) eines Systemdeskriptors enthält in vier Bit kodiert den Typ des Systemdeskriptors. Es beinhaltet außerdem wichtige Informationen für die Zugriffsschutzsicherung.

0-Bytes Die höchstwertigen beiden Bytes jedes Deskriptors werden beim Mikroprozessor 80286 nicht genutzt. Sie sollten beide den Wert 0 haben, um die Softwarekompatibilität zu weiterentwickelten CPU-Typen dieser Mikroprozessorfamilie abzusichern.

Gatedeskriptoren enthalten in insgesamt acht Bytes die folgenden Parameter:



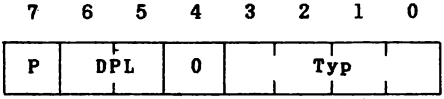
Wortzähler — Anzahl der Worte, die bei einem CALL-Gateübergang aus dem alten Stack in den neuen Stack übertragen werden sollen.

Zielselektor — Auswahl eines Speichersegmentdeskriptors in der Globaldeskriptortabelle (GDT) oder der Lokaldeskriptortabelle (LDT) bzw. eines TSS-Systemdeskriptors in der GDT.

Zieloffset — Auswahl eines Eintrittspunktes in das durch den Zielselektor ausgewählte Speichersegment bei CALL-, Trap- und Interruptgatedeskriptoren.

Der genaue Aufbau des Accesbytes bei Systemdeskriptoren ist den Tafeln 6.4 und 6.5 zu entnehmen

Tafel 6.4. Zusammengefaßte Übersicht aller Systemdeskriptoren des Mikroprozessors 80286 anhand der Identifizierung im Accessbyte



P Präsenzbit (present)

P=0 Das durch den Systemdeskriptor beschriebene Objekt befindet sich nicht im Hauptspeicher.

P=1 Das durch den Systemdeskriptor beschriebene Objekt befindet sich im Hauptspeicher.

Bemerkung: Wenn ein nicht präsentes Objekt angesprochen wird, erfolgt über ein internes Interruptsignal (Interrupt 11) eine Programmunterbrechungsmeldung, die es gestattet, ein nicht im Primärspeicher befindliches Objekt in den Hauptspeicher zu laden und danach den zur Programmunterbrechung führenden Befehl noch einmal auszuführen (Abschn. 9. Interruptsystem).
Das Setzen und Rücksetzen des P-Bits im Accessbyte muß vom Programmierer übernommen werden, es erfolgt nicht automatisch.

DPL Deskriptorprivilegierungsstufe (descriptor privilege level)

DPL=00	Privilegierungsstufe 0
DPL=01	Privilegierungsstufe 1
DPL=10	Privilegierungsstufe 2
DPL=11	Privilegierungsstufe 3

Bemerkung: Die Deskriptorprivilegierungsstufe gibt die Privilegierungsstufe des durch den Deskriptor beschriebenen Objekts an.

Typ Systemdeskriptortyp

=0000	reserviert
=0001	nicht aktives Task-State-Segment TSS (verfügbar)
=0010	LDT-Deskriptor
=0011	aktives Task-State-Segment TSS (besetzt)
=0100	CALL-Gate
=0101	Taskgate
=0110	Interruptgate
=0111	Trapgate
=1000	reserviert
=1001	reserviert
	reserviert
.	
=1111	reserviert

Tafel 6.5. Übersicht über die Kodierung von Systemdeskriptoren (hexadezimal kodiert)

P=0				P=1				Bemerkung
DPL=				DPL=				
0	1	2	3	0	1	2	3	
00	20	40	60	80	A0	C0	E0	reserviert
01	21	41	61	81	A1	C1	E1	TSS-Deskriptor eines verfügbaren TSS
02	22	42	62	82	A2	C2	E2	LDT-Deskriptor
03	23	43	63	83	A3	C3	E3	TSS-Deskriptor eines besetzten TSS
04	24	44	64	84	A4	C4	E4	CALL-Gatedeskriptor
05	25	45	65	85	A5	C5	E5	Taskgatedeskriptor
06	26	46	66	86	A6	C6	E6	Interruptgatedeskriptor
07	27	47	67	87	A7	C7	E7	Trapgatedeskriptor
08	28	48	68	88	A8	C8	E8	reserviert
09	29	49	69	89	A9	C9	E9	reserviert
0A	2A	4A	6A	8A	AA	CA	EA	reserviert
0B	2B	4B	6B	8B	AB	CB	EB	reserviert
0C	2C	4C	6C	8C	AC	CC	EC	reserviert
0D	2D	4D	6D	8D	AD	CD	ED	reserviert
0E	2E	4E	6E	8E	AE	CE	EE	reserviert
0F	2F	4F	6F	8F	AF	CF	EF	reserviert

6.2.3. Deskriptortabellen

Alle Speichersegment- und Systemdeskriptoren eines im PVA-Mode arbeitenden 80286-Mikroprozessorsystems werden in drei Klassen von Deskriptortabellen zusammengefaßt.
Die Deskriptortabellen müssen vom Programmierer bei der Strukturierung und Modularisierung eines 80286-Softwaresystems angelegt werden.

Globaldeskriptortabelle

GDT Die Globaldeskriptortabelle enthält alle Speichersegment- und Systemdeskriptoren von Speichersegmenten, die systemweit gültige Programme oder Daten beinhalten.
Der Zugriff auf Speichersegmente, deren Deskriptoren in der GDT zusammengefaßt sind, ist nur unter Einhaltung von bestimmten Zugriffsschutzrichtlinien möglich.
LDT-Deskriptoren und TSS-Deskriptoren müssen grundsätzlich in die Globaldeskriptortabelle eingeordnet werden.
In einem 80286-Softwaresystem existiert in der Regel eine einzige Globaldeskriptortabelle, die sich ständig im Primärspeicher befindet.

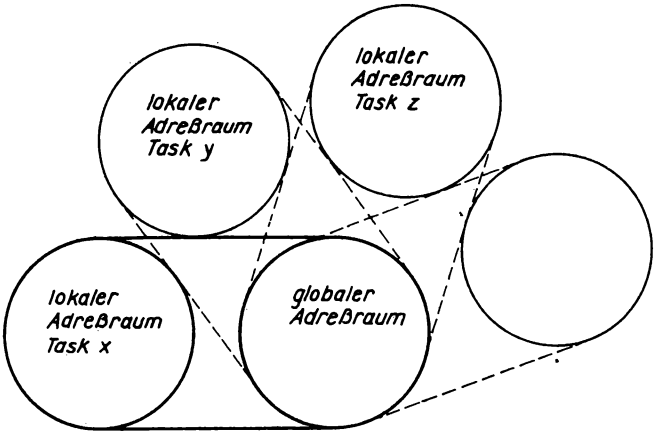
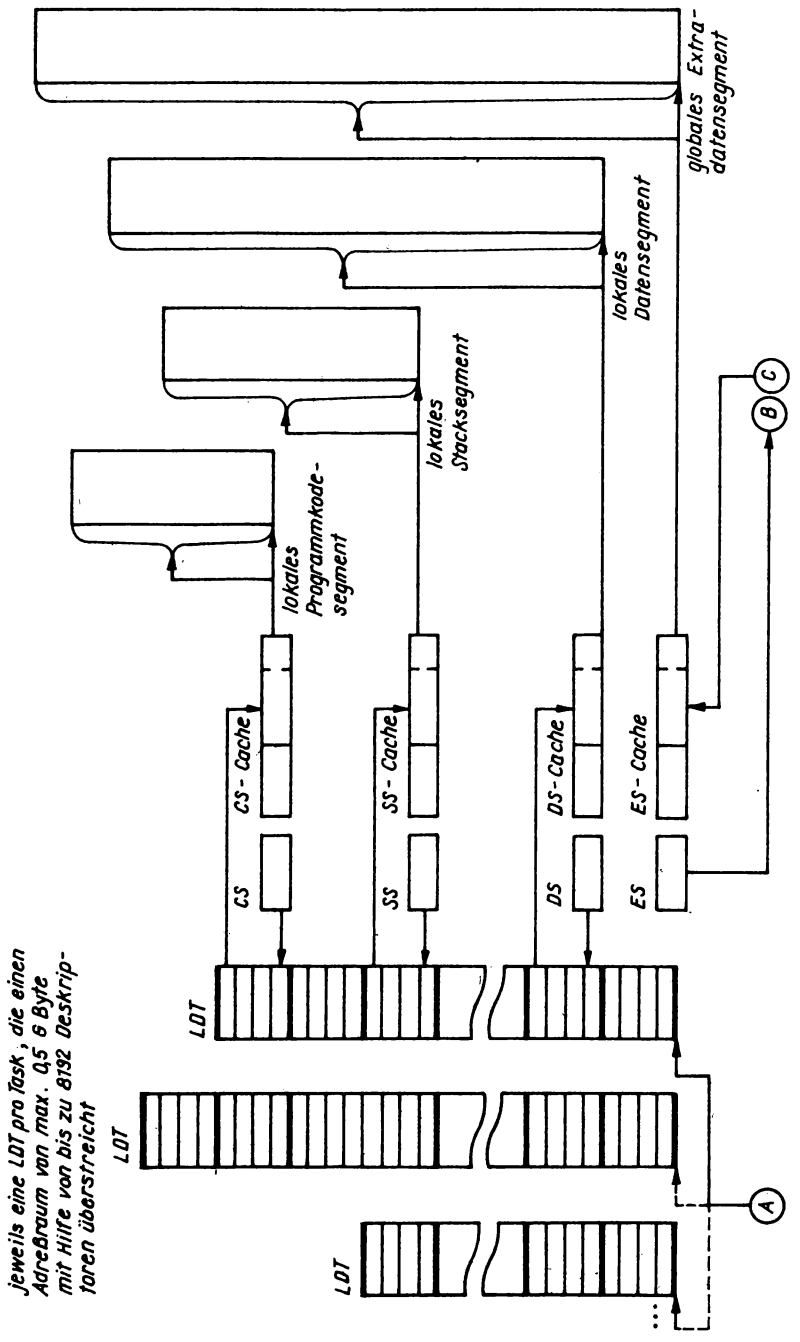


Bild 6.3. Globaler und lokaler Adreßraum einer Task beim Mikroprozessor 80286

Interrupt-deskriptor-tabelle IDT Die Interruptdeskriptortabelle enthält alle Deskriptoren, die den bis zu 256 verschiedenen Interruptserviceroutinen zugeordnet werden.
In der Interruptdeskriptortabelle können sich Interrupt-, Trap- und Taskgatedeskriptoren befinden, in keinem Fall Speichersegmentdeskriptoren.
In einem 80286-Softwaresystem existiert in der Regel eine einzige Interruptdeskriptortabelle, die sich ständig im Primärspeicher befindet.

Lokal-deskriptor-tabelle LDT Je eine Lokaldeskriptortabelle enthält alle Speichersegment- und Gatedeskriptoren, die speziell zu einer Task gehörende Programmkodesegmente, Datensegmente und Gate-Steuerdatenstrukturen beschreibt (Bild 6.3).
Ist eine Task nicht aktiv, kann die Lokaldeskriptortabelle ggf. auch auf einen Hintergrundspeicher ausgelagert worden sein.
Die Speichersegmente, in denen sich Lokaldeskriptortabellen befinden, müssen über LDT-Systemdeskriptoreinträge in der Globaldeskriptortabelle definiert werden.

Die Deskriptortabellen können sich an einer beliebigen Stelle im Speicher eines 80286-Mikrorechnersystems befinden.
Die Reihenfolge der Anordnung der einzelnen Deskriptoren in der Globaldeskriptortabelle (GDT) und in der Lokaldeskriptortabelle (LDT) ist bedeutungslos und frei wählbar.
In der Interruptdeskriptortabelle (IDT) ist die Reihenfolge der Gatedeskriptoreinträge entsprechend der Reihenfolge der bis zu 256 möglichen Interruptsignale vorzunehmen.


jeweils eine LDT pro Task, die einen Adreßraum von max. 0,5 6 Byte mit Hilfe von bis zu 8192 Deskriptoren überstreicht
LDT
LDT
LDT
A
B
C
CS
CS-Cache
SS
SS-Cache
DS
DS-Cache
ES
ES-Cache
lokales Programmkodesegment
lokales Stacksegment
lokales Datensegment
globales Extradatensegment

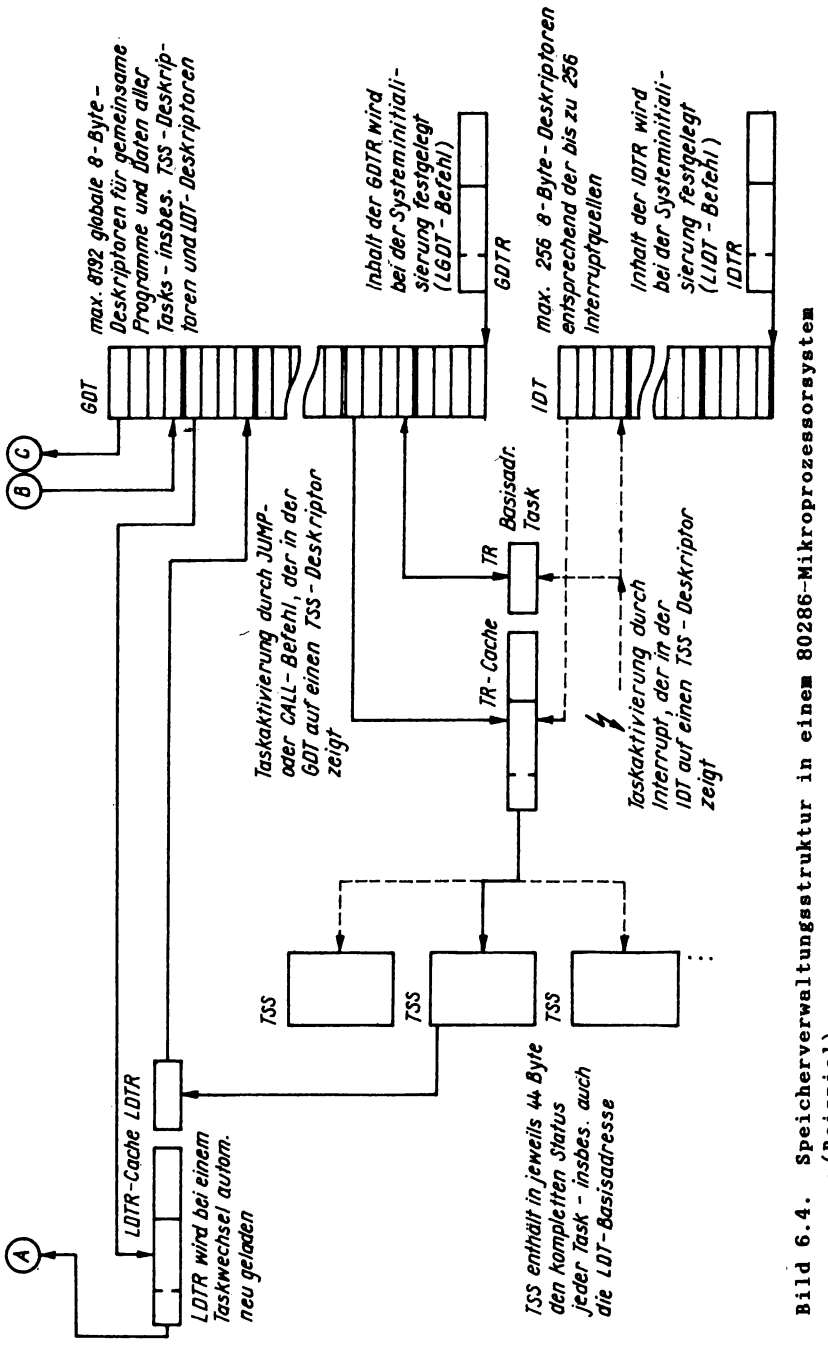


Bild 6.4. Speicherverwaltungsstruktur in einem 80286-Mikroprozessorsystem (Beispiel)

Die Anzahl der möglichen Despriptoreinträge in der Globaldeskriptortabelle (GDT) und der Lokaldeskriptortabelle (LDT) beträgt maximal 8192 - in der Interruptdeskriptortabelle (IDT) maximal 256.
Während der Ausführung einer Task hat diese nur Zugriff auf Speichersegmente, die in ihrer eigenen Lokaldeskriptortabelle zusammengefaßt wurden. Außerdem kann sie, wenn es ihre Zugriffsschutzsrechte erlauben, über die Globaldeskriptortabelle auf Speichersegmente mit globalen Programmen oder Datenstrukturen zugreifen.
Es ist wichtig, an dieser Stelle anzumerken, daß sich - sowohl in der Globaldeskriptortabelle als auch in der Lokaldeskriptortabelle - sogenannte 'leere' Deskriptoreinträge befinden können, die erst im Laufe der Programmabarbeitung genau spezifiziert werden. Voraussetzung für eine solche dynamische Deskriptortabellenbehandlung ist, daß sich die jeweilige Tabelle in einem RAM-Schreib/Lese-Speicherbereich (RAM random access memory) befindet (Abschn. 6.2.4. Alias-Speichersegmente).
Es ist außerdem zu beachten, daß der erste Eintrag in der Globaldeskriptortabelle mit dem INDEX-Wert 0 (Nullselektor) nicht benutzt werden kann. Ein Zugriff führt hier immer zu einem internen Fehlerinterrupt.
Unter Berücksichtigung des Gesagten ergibt sich in einem 80286-Mikroprozessorsystem die im Bild 6.4 an einem Beispiel dargestellte Grundstruktur von Segmentregistern, Systemkonfigurationsregistern, Segmentdeskriptoren, Segmentdeskriptortabellen und Speichersegmenten.

6.2.4. Alias-Speichersegmente

Um den Zugriff auf Speichersegmente beim Mikroprozessor 80286 in unterschiedlicher Weise zu erlauben, existiert die Möglichkeit, ein und dasselbe Speichersegment über mehrere unterschiedliche Speichersegmentdeskriptoren zu definieren, die dann als Alias-Speicherdeskriptoren oder Alias-Speichersegmente bezeichnet werden.
Durch Alias-Speichersegmentdeskriptoren ist es beispielsweise möglich, ein Programmkodesegment als Alias-Datensegment zu definieren und damit indirekt den freizügigen Schreib- und Lesezugriff auf ein Programmkodestück zu erlauben, der sonst nicht gestattet ist.
Genauso ist es möglich, das Speichersegment, in dem sich die Globaldeskriptortabelle (GDT), die Lokaldeskriptortabelle (LDT) und die Interruptdeskriptortabelle (IDT) befinden, als Alias-Datensegment zu definieren und so die gezielte Manipulation in diesen Tabellen dem zu gestatten, dem der Zugriff auf diese Alias-Speichersegmente erlaubt ist (z.B. einem Betriebssystem).
Alle Deskriptortabellen (GDT, LDT und IDT) können sonst weder gelesen noch geschrieben werden.
Alias-Speichersegmente müssen nicht die gleiche Länge besitzen, wie die zu überdeckenden Originalspeicherbereiche. Sie können sowohl größer als auch kleiner definiert werden.
Alias-Speichersegmentdeskriptoren weisen keinerlei Besonderheiten gegenüber anderen Speichersegmentdeskriptoren auf. Sie haben den gleichen Aufbau und die gleiche interne Struktur, wie die im Abschnitt 6.2.1. behandelten Speichersegmentdeskriptoren.

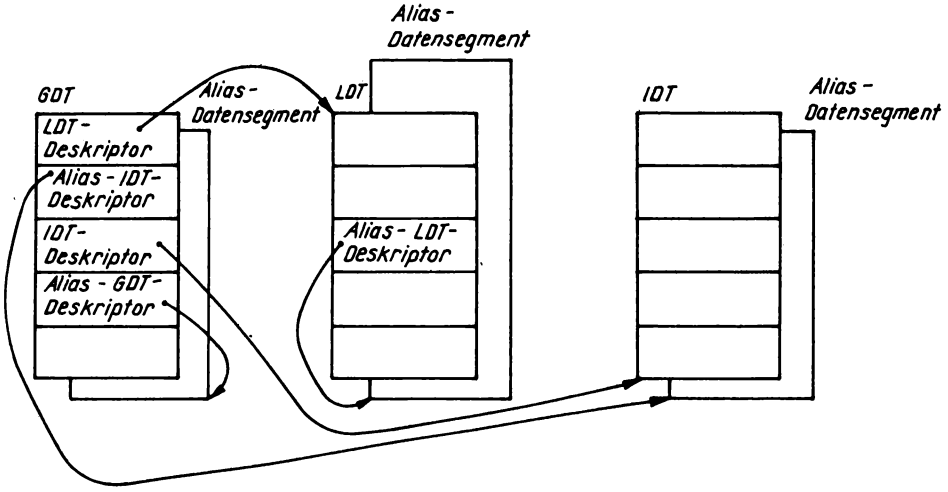


Bild 6.5. Alias-Speichersegmente beim Mikroprozessor 80286 (Beispiel)

7. Zugriffsschutz im PVA-Mode

Zuverlässigkeitsaspekte spielen bei der Entwicklung, Produktion und beim Einsatz von modernen Mikroprozessorsystemen eine entscheidende Rolle. Sie beeinflussen im hohen Maße die Einsatzcharakteristiken eines Gerätes, aber auch die Entwicklungszeit und die Entwicklungskosten.
Untrennbarer Bestandteil der Zuverlässigkeit eines Mikroprozessorsystems sind Softwarefragen und, hier wiederum, die gesamte Problematik des möglichst vollständigen und umfassenden Tests eines Applikationsprogrammsystems.
Ein hundertprozentiger Test eines größeren Programmsystems ist wohl nie möglich. Das gilt um so mehr, als viele Fehler sich erst unter sehr komplexen Testkonstellationen zeigen und, besonders bei Echtzeitsystemen, manchmal außerordentlich schwer exakt zu reproduzieren sind.
Um so bedeutungsvoller sind alle Hardware- und Softwaremaßnahmen, die der Isolierung von Softwarefehlern in einem Mikroprozessorsystem dienen.
In der Entwicklungsphase ermöglichen sie eine schnelle Fehlereingrenzung und Beseitigung. Beim Einsatz schließlich sollen sie sichern, daß sich ein in ein Softwareteilsystem eingeschlichener Fehler möglichst nicht auf die Funktionsfähigkeit des gesamten Gerätesystems auswirkt.
Beim Mikroprozessor 80286 existieren im PVA-Mode (und nur in dieser Betriebsart) mehrere von der Hardware dieses Schaltkreises unterstützte Schutzmechanismen und Architekturmerkmale, die

- der Isolierung der Systemsoftware von den Anwenderprogrammen,
- der Isolierung von Anwenderprogrammen untereinander und
- der Privilegierung bzw. Zugriffsschutzüberprüfung

dienen.
Sie werden alle unter dem Begriff Zugriffsschutz (protection) zusammengefaßt und zeichnen für viele wesentliche Eigenschaften dieses Systems verantwortlich.
Im REAL ADDRESS MODE des Mikroprozessors 80286 existieren keinerlei Zugriffsschutz- oder Privilegierungsmechanismen.

7.1. Basiskonzepte

Bei der Diskussion der im 80286 implementierten Basismechanismen des Zugriffssschutzes spielen die Speichersegmentierung und das Taskkonzept eine herausragende Rolle (Abschnitte 6. Speicherverwaltung und 8. Taskkonzept).
Ein Speichersegment stellt den kleinsten mit einheitlichen Schutzattributen versehenen Speicherbereich für Programm- oder Datenstrukturen dar.
Ein Speichersegment wird im PVA-Mode bezüglich Größe, Lage und Eigenschaften durch den zugehörigen Speichersegmentdeskriptor in einer der beiden möglichen Deskriptortabellen (GDT, LDT) definiert.
Der Zugriff auf ein Speichersegment ist nur in dem durch den Segmentdeskriptor festgeschriebenen Rahmen möglich.
Die für die Speichersegment-Zugriffsüberprüfungen notwendigen Operationen erfolgen von der 80286-Hardware, ohne daß dadurch die Speicherzugriffszeit erhöht wird.

In der Zeit, in der die Adreßeinheit des 80286 den Adreßoffset zur Segmentanfangsadresse addiert, wird gleichzeitig überprüft, ob der Adreßoffset nicht größer ist als die im Segmentdeskriptor abgelegte Segmentgröße. Außerdem finden Überprüfungen statt, die feststellen, ob der eingeleitete Speicherzugriff mit dem Segmenttyp (Programmkode, Daten, Stack) und mit den im Accessbyte vereinbarten Eigenschaften (read, write) konform geht.
Dadurch wird beispielsweise verhindert, daß in ein Programmkodesegment geschrieben wird oder daß auf ein Datensegment ein Befehlslesezugriff erfolgt.
Neben diesen Überprüfungen, die bei jedem Speicherzugriff erfolgen, wird vom Mikroprozessor 80286 beim Laden eines der vier Segmentregister automatisch überwacht, daß der geladene Speichersegmentdeskriptor zur Art des Segmentregisters paßt, also kein 'Read-only-Stacksegment' oder kein 'Programmkode-Datensegment' entsteht.
Beim Erkennen von solchen Zugriffsfehlern wird ein interner Softwareinterrupt ausgelöst, der über eine Fehlerbehandlungsroutine ausgewertet werden kann (Abschn. 9. Interruptsystem).
Alle eben beschriebenen Zugriffsüberprüfungen erfolgen im Mikroprozessor 80286, unabhängig von speziellen Softwarestrukturen, bei jedem Speicherzugriff.
Sie ermöglichen den Speicherzugriffsschutz durch die Verhinderung unzulässiger Operationen. Sie gestatten aber nicht den Schutz des Betriebssystems gegenüber den Anwenderprogrammen oder den Schutz der Anwenderprogramme untereinander.
Um auch diese Forderung erfüllen zu können, wurden im Mikroprozessor 80286 mehrere Hardware/Software-Verriegelungsmechanismen implementiert.
Sie gestatten

- die Aufteilung des gesamten verfügbaren Speicherbereichs in zwei gegeneinander geschützte Adreßräume - einen globalen und einen lokalen Adreßraum (Abschnitte 5. Adreßraum und 6. Speicherverwaltung),

- die Strukturierung eines Softwaresystems in mehrere gegeneinander gesicherte Tasks (Abschn. 8. Taskkonzept) und

- die programmgesteuerte Zuordnung einer von insgesamt vier Privilegierungsstufen (0, 1, 2 oder 3) zu jedem Programm-, Daten- oder Stackspeichersegment.

Die Mechanismen der Zuordnung einer Privilegierungsstufe zu einem Programm-, Daten- oder Stacksegment sollen hier genauer untersucht werden. Sie sichern den in Abarbeitung befindlichen Programmen unterschiedliche Möglichkeiten beim Zugriff auf andere Programme und Datenstrukturen.
Die Privilegierungsstufe 0 besitzt die höchsten, die Privilegierungsstufe 3 die niedrigsten Privilegien.
Privilegierungsstufen können statisch oder dynamisch angewendet werden.
In der Privilegierungsstufe 0 wird wohl immer der Kern des Betriebssystems arbeiten. Er übernimmt solche Aufgaben wie die Prozeßsteuerung, die Prozeßsynchronisation, die Intertaskkommunikation, die Verwaltung der Systemressourcen u.a.m.

Der Kern eines Betriebssystems muß unantastbar gegenüber allen anderen Programmen arbeiten können.
In der Privilegierungsstufe 1 können zentrale Dienste des Betriebssystems laufen, wie beispielsweise die Ein-/Ausgabe-Abwicklung, Swappingroutinen usw.
In der Privilegierungsstufe 2 können mehr oder weniger zentrale Bibliotheksfunktionen od.a. untergebracht werden.

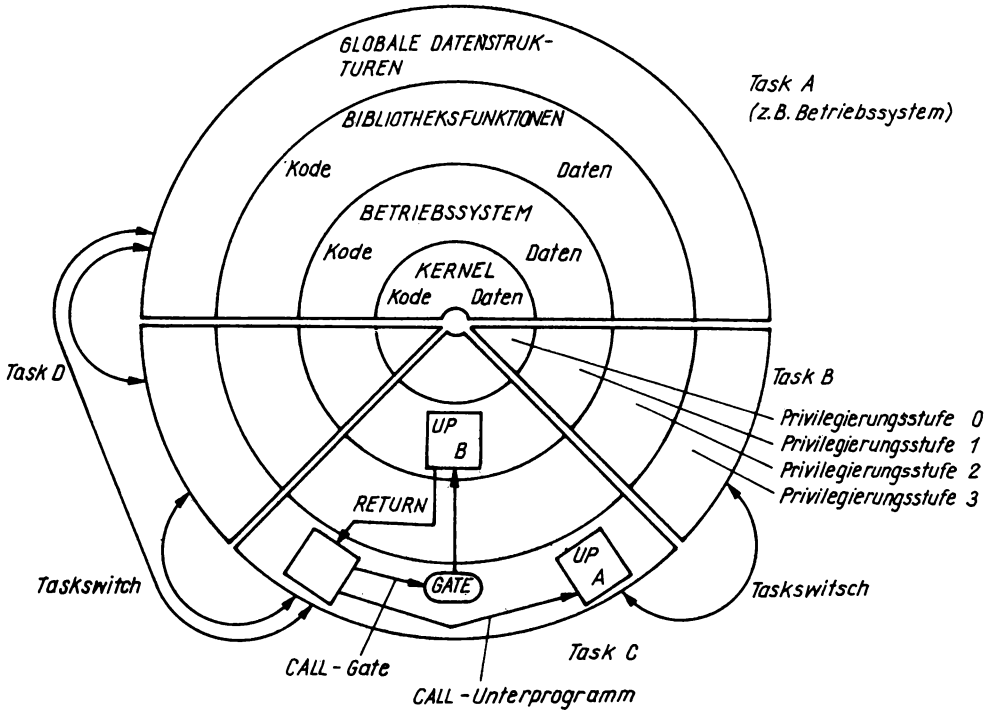


Bild 7.1. Beispiel für die Nutzung der vier Privilegierungsstufen des Mikroprozessors 80286

Die Privilegierungsstufe 3 schließlich kann beispielsweise den Anwenderprogrammen vorbehalten bleiben.
Zu beachten ist, daß nicht in jedem Fall alle vier Privilegierungsstufen genutzt werden müssen, so daß auch klassische zweistufige Privilegierungssysteme (Betriebssystem - Anwenderprogramme) mit dem Mikroprozessor 80286 verwirklicht werden können.

7.2. Privilegierungsstufen

Bei der Behandlung der Priviligierungsstufen im PVA-Mode des Mikroprozessors 80286 werden folgenden Begriffe verwendet:

DPL descriptor privilege level — Die Deskriptorprivilegierungsstufe (DPL) eines Speichersegments wird durch die im Accessbyte (Bit 5 und 6) des Speichersegmentdeskriptors eingetragene Deskriptorprivilegierungsstufe (DPL) festgelegt.
Sie kann statisch festgeschrieben werden oder im Laufe der Programmabarbeitung durch Manipulation des Speichersegmentdeskriptors eine Veränderung erfahren (Abschn. 6.2.1. Speichersegmentdeskriptoren).

CPL current privilege level — Die aktuelle Privilegierungsstufe (CPL) ist die Privilegierungsstufe des Programmkodesegments, das in einem bestimmten Moment gerade abgearbeitet wird (Ausnahme: Conformingsegment).
Der aktuelle CPL-Wert wird in den Bits 0 und 1 des Kodesegmentregisters (CS) innerhalb des RPL-Feldes abgelegt.
Die aktuelle Privilegierungsstufe (CPL) entspricht der Deskriptorprivilegierungsstufe (DPL) des Speichersegmentdeskriptors, der das Speichersegment spezifiziert, in dem sich das momentan ablaufende Programm befindet.

RPL requested privilege level — Die geforderte Privilegierungsstufe (RPL) ist in den Bits 0 und 1 des 16-Bit-Segmentselektors eines Datenspeichersegments kodiert.
Für den Zugriff auf ein Datenspeichersegment muß gelten:

$$CPL <= RPL <= DPL.$$

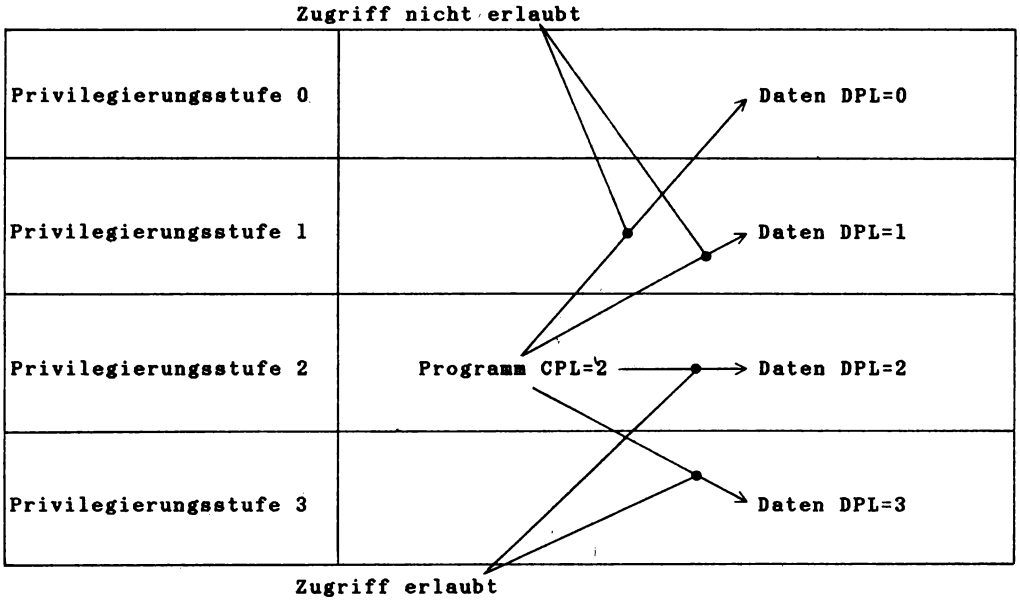
EPL effective privilege level — Die effektive Privilegierungsstufe (EPL) entspricht dem numerischen Maximalwert von aktueller Privilegierungsstufe (CPL) und geforderter Privilegierungsstufe (RPL).

Die aktuelle Privilegierungsstufe (CPL) einer laufenden Task kennzeichnet die ihr zu einem bestimmten Zeitpunkt zugeordneten Zugriffsprivilegien. Die aktuelle Privilegierungsstufe (CPL) ist, wie schon erläutert, die Privilegierungsstufe des Programmkodesegments, das im Moment gerade bearbeitet wird.
Für den Zugriff auf Datenspeichersegmente von einem Programm aus gilt die Vorschrift, daß nur dann zugegriffen werden darf, wenn die aktuelle Privilegierungsstufe (CPL) des in Abarbeitung befindlichen Programms numerisch kleiner oder gleich der Deskriptorprivilegierungsstufe (DPL) des

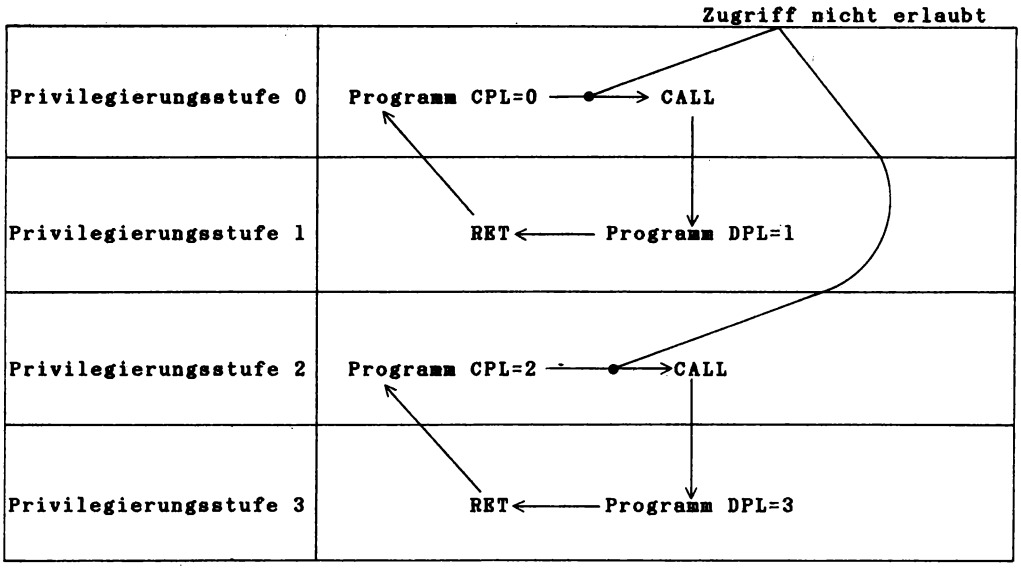
Datenspeichersegments ist, auf das zugegriffen werden soll.
Es kann also nur auf Datensegmente zugegriffen werden, deren Privilegierung gleich oder niedriger als die Privilegierung des momentan ausgeführten Programms ist.
Bei Datenzugriffen auf Datensegmente muß immer gelten

$$RPL = DPL <= CPL,$$

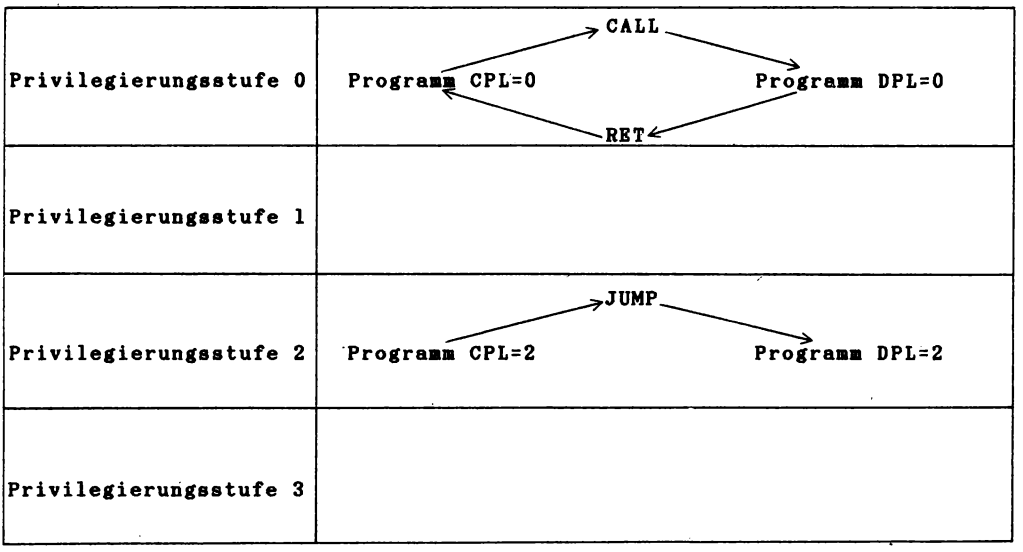
anderenfalls erfolgt eine Fehlermeldung über einen internen Softwareinterrupt.
Ein Zugriff von einem Programm mit einer niedrigen Privilegierungsstufe auf ein Datensegment mit einer höheren Privilegierungsstufe wird generell ausgeschlossen.



Soll innerhalb einer Task von einem bestimmten Programm ein anderes Programm in einem anderen Programmspeichersegment aufgerufen werden (JUMP-/CALL-Befehle), gilt grundsätzlich, daß niemals ein Programm erreicht werden kann, dessen Privilegierungsstufe geringer ist. Dieser Fall wird beim Mikroprozessor 80286 prinzipiell ausgeschlossen, so daß Programme innerhalb einer Task niemals die Privilegierung verschlechtern können. Der Zugriff auf Programme in Programmkodesegmenten mit schlechterer Privilegierungsstufe wird ein für allemal verwehrt.

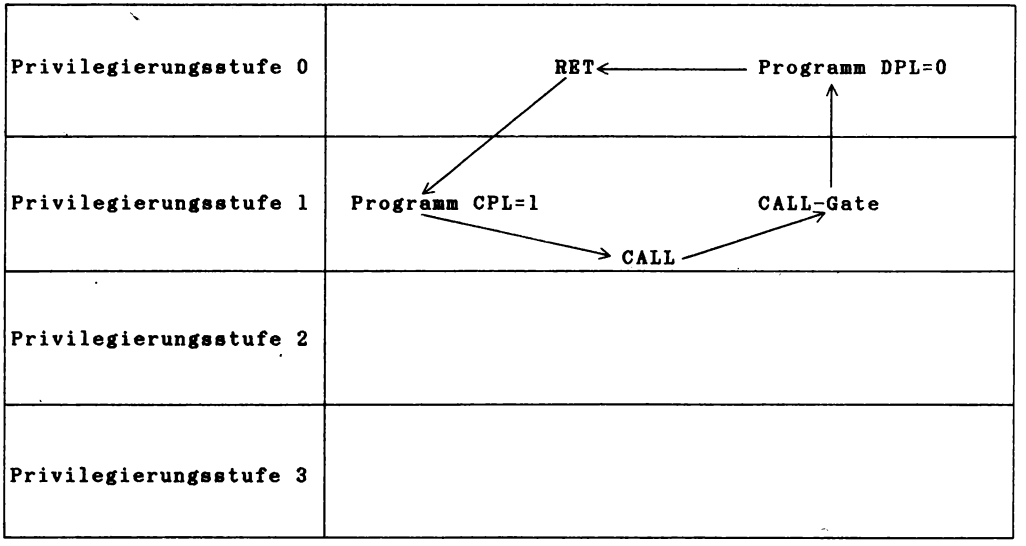


Für Zugriffsmöglichkeiten auf Programme in Programmkodesegmenten in der gleichen Privilegierungsstufe existieren keinerlei Einschränkungen.



Soll die aktuelle Privilegierungsstufe zu einer höheren Privilegierung hin verändert werden, muß in der laufenden Task von dem momentan abgearbeiteten Programm eine neues Programm mit höherer Privilegierungsstufe nach einem streng festgelegten Verfahren aufgerufen werden.

Die von der Hardware des Mikroprozessors 80286 für diese Privilegierungserhöhung vorgeschriebenen Mechanismen bauen auf den im Abschnitt 6. (Speicherverwaltung) bereits erwähnten CALL-Gatedeskriptoren auf (siehe auch Bild 6.2).
CALL-Gatedeskriptoren gestatten den Übergang von Programmen mit einer niedrigen aktuellen Privilegierungsstufe (CPL) zu Programmen mit einer höheren aktuellen Privilegierungsstufe (CPL) und zurück unter genau definierten, von der Hardware des 80286 überwachten Bedingungen.
Auf diese Weise kann beispielsweise ein Anwenderprogramm mit niedriger CPL eine Betriebssystemroutine mit hoher CPL nutzen. Diese Veränderung, hin zu einer höheren Privilegierungsstufe, bezieht sich ausschließlich auf Programmkodesegmente.



Es existieren also drei Möglichkeiten innerhalb einer Task, die Steuerung von einer Routine auf eine andere Routine zu übergeben:

- innerhalb eines Programmkodesegments (intrasegment) ohne Veränderung der Privilegierungsstufe mit einem der Befehle

  JMP(near), CALL(near)/RET,

- zwischen verschiedenen Programmkodesegmenten (intersegment) der gleichen Privilegierungsstufe

  JMP(far), CALL(far)/RET,

- zwischen zwei Segmenten (intersegment) mit unterschiedlicher Privilegierungsstufe über die Vermittlung eines CALL-Gatedeskriptors

  CALL(far)/RET.

In den ersten beiden Fällen existieren keine gesondert zu beachtenden Einschränkungen bezüglich der Zugriffsrechte. Im dritten Fall muß ein Übergang von einer niedrigen Privilegierungsstufe zu einer höheren Privilegierungsstufe generell über einen CALL-Gatedeskriptor erfolgen.
Ein umgekehrter Wechsel von einer höheren Privilegierungsstufe zu einer niedrigeren Privilegierunsstufe ist nicht möglich.

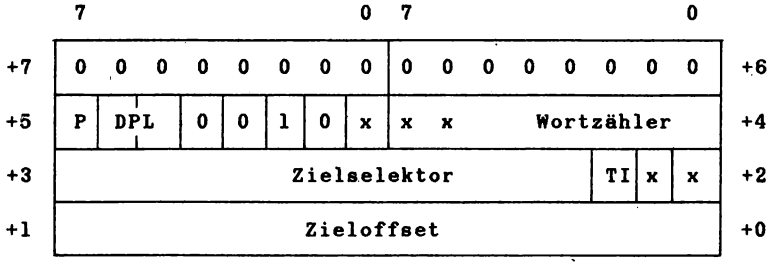
7.3. CALL-Gatedeskriptor — Privilegierungserhöhung

Ein CALL-Gatedeskriptor definiert

- das Speichersegment, in dem sich die aufrufende Routine befindet,
- die Startadresse der aufgerufenen Routine und
- die Anzahl der 16-Bit-Worte, die als Aufrufparameter vom Stack der aufrufenden Routine in den Stack der aufgerufenen Routine automatisch (hardwaregesteuert) kopiert werden sollen (0 bis maximal 31 Worte).

CALL-Gatedeskriptoren können sich in der Globaldeskriptortabelle (GDT) und in der Lokaldeskriptortabelle (LDT) befinden.
Der Aufruf eines CALL-Gatedeskriptors erfolgt - der Name sagt es schon - über einen CALL-Befehl. Der im CALL-Befehl angegebene Segmentselektor wählt in der GDT- oder LDT-Deskriptortabelle das spezifizierte CALL-Gate aus. Der im CALL-Befehl angegebene Offsetwert wird ignoriert. Die Startadresse wird dem Gate selbst entnommen und nicht dem CALL-Befehl. Nach der Ausführung der höherprivilegierten Routine erfolgt die Rückkehr in das aufrufende Programm über einen RET-Befehl.
Das Gatekonzept erlaubt einen programmtechnisch sauberen, sicheren und, wegen der extrem kurzen Ausführungszeit des CALL-Befehls, auch effizienten Aufruf von höherprivilegierten Programmen (z.B. Systemdiensten) aus niedriger privilegierten Routinen (z.B. Anwenderprogrammen).
Der Aufruf von Programmen innerhalb eines aktuellen Speichersegments erfolgt grundsätzlich nicht über Gatedeskriptoren, sondern direkt mit JUMP(near)-, CALL(near)- und RET-Befehlen.
Soll ein Programm außerhalb des aktuellen Speichersegments auf der gleichen Privilegierungsstufe aufgerufen werden, kann der Programmierer zwischen einem CALL-Gateübergang und einem direkten Aufruf auswählen.
Bei einem direkten Aufruf wird zwar der gleiche CALL-Befehl des 80286 wie bei einem CALL-Gateübergang genutzt, der Segmentselektor zeigt aber in diesem Fall nicht auf einen Gatedeskriptor, sondern auf einen Programmkode-Segmentdeskriptor. Der im Befehl angegebene Offsetwert stellt dann direkt die Startadresse der aufzurufenden Routine dar.
Der CALL-Gateübergang zu einem neuen Programmkodesegment auf der gleichen Privilegierungsstufe sollte z.B. immer dann verwendet werden, wenn das CALL-Gate sicherstellen soll, daß bestimmte Programme - einheitlich von unterschiedlichen Ausgangspunkten aus - nur über die Vermittlung eben eines CALL-Gates aufgerufen werden.

CALL-Gatedeskriptoren enthalten in insgesamt acht Bytes die folgenden Parameter:



Wortzähler	Anzahl der Worte, die bei einem CALL-Gateübergang automatisch aus dem alten Stack in den neuen Stack übertragen werden sollen (0 bis 31 Worte).
Zielselektor	Auswahl eines Speichersegmentdeskriptors in der Globaldeskriptortabelle (GDT) oder der Lokaldeskriptortabelle (LDT), in dem sich das mit dem CALL-Gatedeskriptor aufzurufende Programm befindet.
Zieloffset	Auswahl eines Eintrittspunktes in dem durch den Zielselektor ausgewählten Speichersegment.
P	Siehe Tafeln 6.4 und 6.5.
x	Nicht genutzt.

Für die Zugriffserlaubnis von einem Programm aus auf einen CALL-Gatedeskriptor ergibt sich die Beziehung

$$CPL \leq DPL$$

des CALL-Gate (DPL=RPL).
Das bedeutet, daß das aufrufende Programm entweder die gleiche oder eine höhere Privilegierungsstufe haben muß als den DPL-Wert im CALL-Gatedeskriptor. Es gelten also beim Zugriff auf einen CALL-Gatedeskriptor die gleichen Bedingungen wie beim Zugriff auf Datensegmente.

7.4. CALL-Gatedeskriptor — Statuswechsel

Um Programme mit unterschiedlichen Privilegierungsstufen innerhalb einer Task möglichst weitgehend voneinander zu separieren und gegeneinander zu schützen, wird durch die Systemarchitektur des 80286 gesichert, daß jede Privilegierungsstufe einer Task einen eigenen Stackspeicherbereich besitzt.

Beim Wechsel der Privilegierungsstufe über einen Gatedeskriptor ist deshalb zu beachten, daß ein automatischer Wechsel des verwendeten Stackspeicherbereichs erfolgt. Dadurch wird abgesichert, daß jede Privilegierungsstufe einer Task ihren eigenen Stackspeicherbereich besitzt. Die Stackpointer der unterschiedlichen Privilegierungsstufen werden dem bei der Programminitialisierung anzulegenden Task-State-Segment (TSS) entnommen.
Bei den im Task-State-Segment abgelegten Stackpointern für die unterschiedlichen Privilegierungsstufen ist zu beachten, daß für die Privilegierungsstufe 3 kein zusätzlich vordefinierter Stackpointerinitialwert benötigt wird, da Programme in dieser Privilegierungsstufe nie von Programmen auf einer anderen Privilegierungsstufe aufgerufen werden können (Abschn. 8. Taskkonzept).
Die Möglichkeit, bei einem CALL-Gateübergang automatisch bis zu 31 Worte (Wortzählerfeld im CALL-Gate) aus dem alten Stack des Programms mit der niedrigen Privilegierungsstufe in den neuen Stack des Programms mit der höheren Privilegierungsstufe zu kopieren, sichert die problemlose hardwaregesteuerte Parameterübergabe.
Der Programmierer hat dabei natürlich die Stackspeicherbereichskalkulation vorher möglichst genau vorzunehmen. Sollte der neue Stackdatenbereich für die zu übernehmenden Parameter nicht ausreichen, erfolgt ein interner Softwareinterrupt, der diesen Fehler anzeigt.

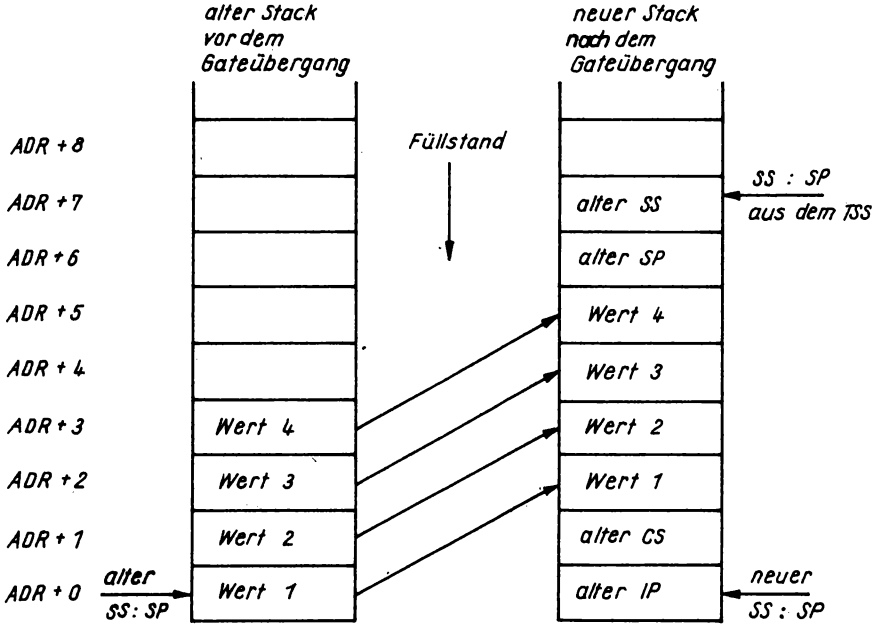


Bild 7.2. Stackdatenbereiche bei einem Gatedeskriptorübergang (CALL-Gate) in einem 80286-Mikroprozessorsystem

8. Taskkonzept PVA-Mode

Eine Task ist eine definierte, in Abarbeitung befindliche, abgegrenzte Teilkomponente eines großen Programmsystems mit allen die Programmabarbeitung spezifizierenden Informationen, wie Programmkode, Datenstrukturen, Registerinhalte, Stackspeicherinhalte, Statuszustände u.a.m.
Eine Task kann aus einem oder mehreren einzelnen Programmen bestehen, die miteinander durch wechselseitige Aufrufe in Verbindung stehen.
Tasks werden verwendet, um die Arbeit eines aus mehreren, weitestgehend gegeneinander isolierten Teilkomponenten - den einzelnen Tasks - bestehenden Gesamtsystems leichter und effektiver verstehen, programmieren, testen und pflegen zu können.
Klassische Anwendungsfälle von Multi-Task-Systemen sind Echtzeitbetriebssysteme und Time-sharing-/Multi-user-Betriebssysteme.
Es ist wichtig, an dieser Stelle hervorzuheben, daß alle in der Folge beschriebenen Eigenschaften des Mikroprozessors 80286 bezüglich hardwaregestützter Taskstrukturen nur im PROTECTED VIRTUAL ADDRESS MODE zur Verfügung stehen.

8.1. Taskwechselmechanismen

Da in einem sequentiell arbeitenden 80286-Mikroprozessorsystem gleichzeitig immer nur eine bestimmte Task von in der Regel mehreren verfügbaren Tasks bearbeitet werden kann, spielen Fragen der schnellen und gesicherten Umschaltung von einer laufenden Task auf eine andere (task switch) eine außerordentlich wichtige Rolle.
Beim Mikroprozessorsystem 80286 existieren, bezogen auf die Taskwechselmechanismen, im PVA-Mode die folgenden beiden hervorzuhebenden Merkmale:

- von der Hardware des 80286 voll unterstützte außerordentlich schnelle Taskumschaltung mit vollständigem Statuswechsel in 22 Mikrosekunden (8 MHz 80286-CPU),

- Schutz jeder Task gegenüber einem Zugriff von einer anderen Task durch umfassende Separierung ihres zugehörigen Adreßraums und Statuszustands (Registerinhalte, Stackspeicherbereiche usw.).

Eine Task wird in einem 80286-Mikroprozessorsystem immer durch eine spezielle 44 Byte lange Steuerdatenstruktur, das sogenannte Task-State-Segment (TSS), definiert. Das Task-State-Segment stellt einen besonderen, nur dem Taskkonzept des 80286 zugeordneten Systemspeichersegmenttyp dar.
Für jede Task in einem 80286-Softwaresystem existiert jeweils ein ganz spezielles, dieser Task zugeordnetes Task-State-Segment, das vom Programmierer in dem dafür vorgesehenen Systemspeichersegment angelegt wird.
Die Aktivierung einer Task bei der ersten Initialisierung eines Programmsystems (Urtask) oder beim Taskwechsel kann über folgende Befehle erfolgen:

- JMP-Befehl (Taskwechsel-JMP),
- CALL-Befehl (Taskwechsel-CALL),
- Interrupt (Taskwechselinterrupt),
- IRET-Befehl (Taskwechsel-RETURN).

Ein JMP- oder CALL-Befehl, der einen Taskwechsel auslösen soll, muß den zu einem Task-State-Segment gehörenden TSS-Deskriptor über einen im Befehl kodierten TSS-Segmentselektor in der Globaldeskriptortabelle (GDT) ansprechen.
Ein Interrupt, der einen Taskwechsel auslösen soll, muß mit seinem Interruptvektor in der Interruptdeskriptortabelle (IDT) auf einen Taskgatedeskriptor verweisen. Der Taskgatedeskriptor stellt indirekt den Bezug auf einen TSS-Deskriptor her.
Hier ist anzumerken, daß Interrupts nicht unbedingt einen Taskwechsel auslösen müssen. Der Interruptvektor kann auch auf eine Routine zeigen, die innerhalb der laufenden Task ausgeführt wird. Der interruptgesteuerte Taskwechsel wird im Abschnitt 8.7. (Taskgatedeskriptor) genauer untersucht.
Die Taskwechselmechanismen selbst können sich von Fall zu Fall, entsprechend den in der Folge noch genau zu beschreibenden Möglichkeiten beim 80286, etwas voneinander unterscheiden (Bild 8.1).

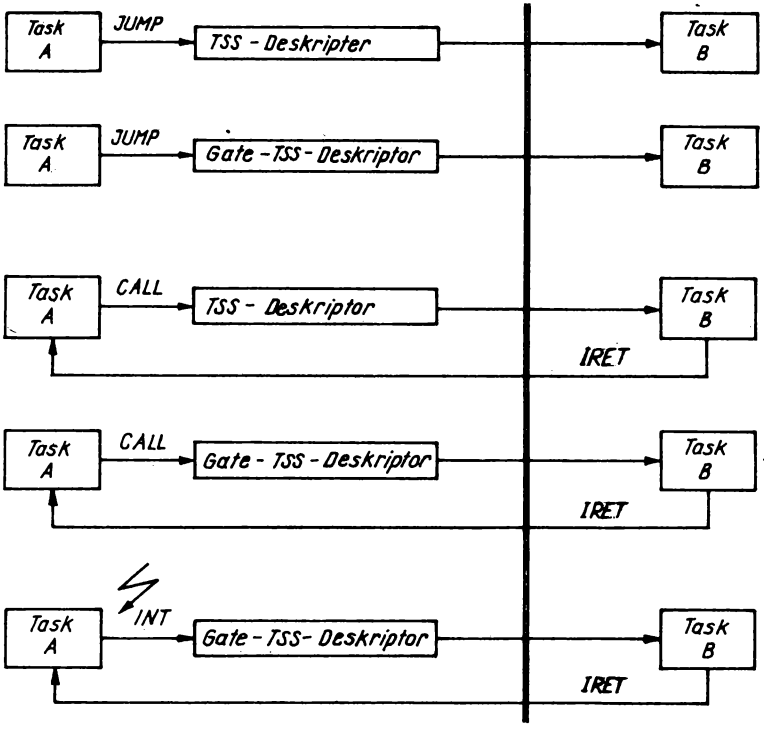


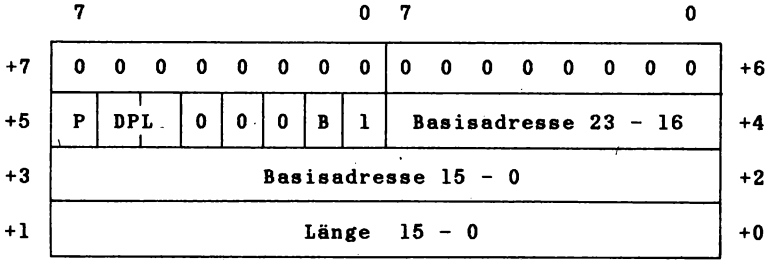
Bild 8.1. Taskwechselmechanismen beim Mikroprozessor 80286

8.2. TSS-Deskriptor

Ein zu einem Task-State-Segment gehörender TSS-Deskriptor muß sich ständig zugriffsbereit in der Globaldeskriptortabelle (GDT) im Primärspeicher des 80286-Mikrorechners befinden.
Ein TSS-Deskriptor ist durch die Bitbelegung

000B1 B=0 TSS frei - Task nicht aktiv (idle)
oder
000B1 B=1 TSS besetzt - Task läuft (busy)

in den Positionen 4, 3, 2, 1 und 0 im Accessbyte des TSS-Deskriptors gekennzeichnet.
Ein TSS-Deskriptor enthält in insgesamt acht Bytes folgende Parameter (Abschn. 6.2.2. Systemdeskriptoren, siehe auch Bild 6.2 und Tafel 6.4 bzw. auch 6.5):



Länge — Die Länge eines TSS-Deskriptors wird als 16-Bit-Wert angegeben.

Basisadresse — Die 24-Bit-Basisadresse ist bei TSS-Deskriptoren die physische Anfangsadresse eines TSS-Speichersegments in einem 16 MByte großen physischen Adreßraum.

Genauso wie bei einem Programmkode- oder Datensegmentdeskriptor beinhaltet der TSS-Deskriptor ein Basisadressen- und ein Längenfeld.
Die Basisadresse spezifiziert die genaue Anfangsadresse des Task-State-Segments.
Die Längenangabe muß - resultierend aus der festgeschriebenen Größe eines TSS-Segments - 43 (002BH) betragen. Anderenfalls erfolgt eine interne Fehlerinterruptmeldung beim ersten Zugriff.
Das P-Bit (present) im Accessbyte des TSS-Deskriptors kennzeichnet, ob die im TSS-Deskriptor im Moment eingetragenen Informationen gültig sind (P=1) oder nicht (P=0). Ein versuchter Taskwechsel über einen ungültigen TSS-Deskriptor (P=0) führt immer zu einem internen Fehlerinterrupt.

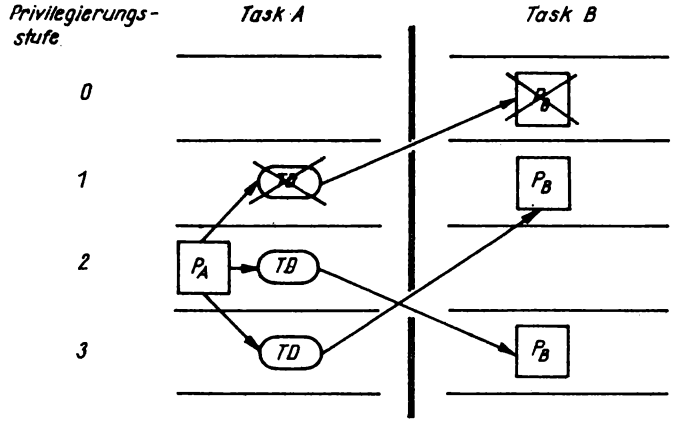


Bild 8.2. Gültige und ungültige Taskwechselversuche beim Mikroprozessor 80286 (PA Programmkode Task A; PB Programmkode Task B; TD TSS-Deskriptor Task B)

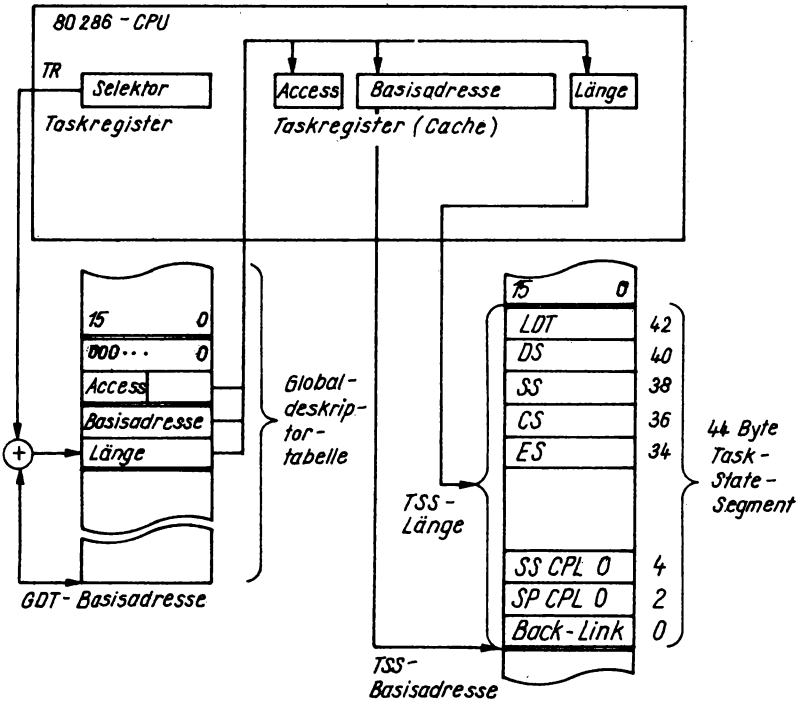


Bild 8.3. Zusammenhang zwischen TSS-Segmentselektor, TSS-Segmentdeskriptor und Task-State-Segment beim Mikroprozessor 80286

Das B-Bit (busy) im Accessbyte des TSS-Deskriptors kennzeichnet, ob die zum TSS-Deskriptor gehörende Task im Moment bearbeitet wird, (B=1) also aktiv ist oder ob die Task inaktiv auf einen Aufruf über einen Taskwechsel wartet (B=0).
Das B-Bit im TSS-Deskriptor wird beim Aufruf einer Task von der 80286-CPU automatisch gesetzt bzw. in der austretenden Task rückgesetzt.
Die im Accessbyte eingetragene Privilegierungsstufe (DPL) wird, entsprechend den im Abschnitt 7. (Zugriffsschutz) gemachten Ausführungen zur Zugriffssteuerung auf das TSS-Segment, über die einen Taskwechsel auslösenden JMP- bzw. CALL-Befehle verwendet.
Dabei gilt grundsätzlich, daß ein Taskwechsel in einem 80286-Mikroprozessorsystem nur dann ausgeführt wird, wenn die aktuelle Privilegierungsstufe (CPL) des Programms mit dem aufrufenden JMP- oder CALL-Befehl numerisch kleiner oder gleich der im Accessbyte eines TSS-Deskriptors abgelegten Privilegierungsstufe (DPL) des Task-Status-Segments ist.
Vorteilhaft ist es deshalb immer, eine hohe Privilegierungsstufe zu kodieren (z.B. DPL=00), um nichtautorisierten Programmen außerhalb des Betriebssystems die Möglichkeit zu nehmen, einen Taskwechsel zu initieren.
Fehlerhafte Zugriffe hinsichtlich der Privilegierungsstufe des TSS-Deskriptors führen, ähnlich wie bei einem Zugriff auf einen ungültigen TSS-Deskriptor (P=0) oder wie bei einem Zugriff auf ein besetztes Task-State-Segment (B=1), zu internen Fehlerinterruptmeldungen.
Es ist wichtig, an dieser Stelle anzumerken, daß die Privilegierungsstufe der aufgerufenen Task von der Privilegierungsstufe der aufrufenden Task nicht eingeschränkt wird, da die Privilegierungsstufe der aufgerufenen Task nicht mit der Privilegierungsstufe des zu dieser Task gehörenden TSS-Deskriptors übereinstimmen muß. Die Privilegierungsstufe der neuen Task wird letztlich indirekt immer über die im Task-State-Segment abgelegten Parameter festgelegt (Bild 8.2).

8.3. Taskregister

Die in sich geschlossene Informationskette

- TSS-Segmentselektor,
- TSS-Segmentdeskriptor in der GDT,
- Task-State-Segment

identifiziert eine Task in einem 80286-Mikroprozessorsystem eindeutig.
Der TSS-Segmentselektor, der in der Globaldeskriptortabelle (GDT) den TSS-Deskriptor der aktuell laufenden Task auswählt, wird im Taskregister (TR) des 80286 abgelegt (Bilder 6.4 und 8.3).
Das Taskregister TR in der 80286-CPU besteht, wie auch das LDT-Register, immer aus zwei Teilen:

- einem durch ein Programm direkt beeinflußbaren 'sichtbaren' Teil mit dem eigentlichen TSS-Segmentselektor und

- einem 'unsichtbaren' Cache-Teil, der die informationstragenden Teile des TSS-Deskriptors (TSS-Basisadresse, TSS-Länge, TSS-Accessbyte) beinhaltet.

Der Cache-Teil des TR-Registers wird von der 80286-CPU automatisch bei jeder Veränderung des direkt beeinflußbaren 'sichtbaren' TR-Registerteils aktualisiert.

8.4. Task-State-Segment

Ein Task-State-Segment besteht immer aus einer 44 Byte langen, einheitlich aufgebauten Datenstruktur.
Es definiert den Inhalt

- einer Reihe von CPU-Registern,
- der Flags,
- der Stackpointer der Privilegierungsstufen 0, 1 und 2,
- des Segmentselektors der zu der Task gehörenden Lokaldeskriptortabelle (LDT) sowie
- einen Verweis auf die vorher ausgeführte Task im sogenannten Back-Link-Feld (Bild 8.4).

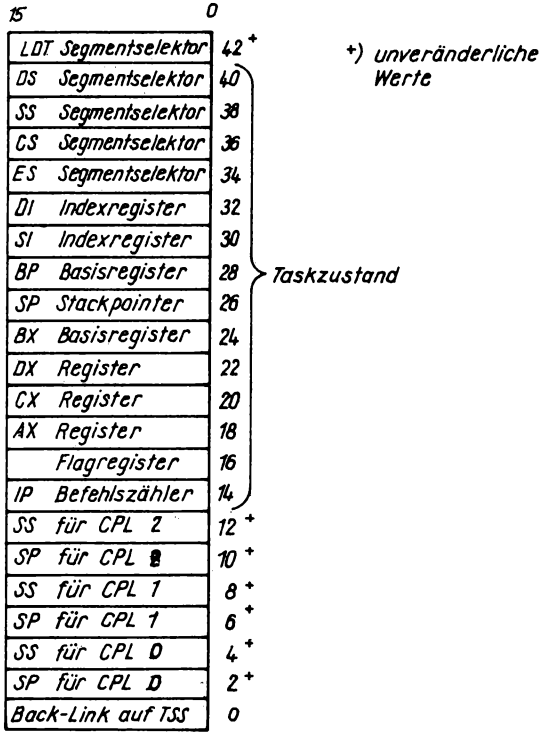


Bild 8.4. Aufbau des Task-State-Segments beim Mikroprozessor 80286

Da jedes Task-State-Segment sowohl unveränderliche Daten enthält (LDT-Segmentselektor, Initialwerte der Stackpointer) als auch Daten beinhaltet, die sich dynamisch, in Abhängigkeit von dem jeweiligen Taskaufruf, verändern (Register, Flags), muß es sich grundsätzlich in einem RAM-Speicherbereich befinden.
Sowohl bei der Aktivierung einer Task über einen CALL- oder JMP-Befehl als auch bei der Aktivierung über einen Interrupt rettet der Mikroprozessor 80286 automatisch den Basisregistersatz der laufenden Task in deren Task-State-Segment, um anschließend alle Registerinhalte der neuen Task aus dem neuen Task-State-Segment zurückzuspeichern.
Danach erfolgt die Umschaltung auf die neue Lokaldeskriptortabelle und das Laden aller Speicherdeskriptoren in die CPU-Segmentregister.
Zum Schluß wird in das neue Task-State-Segment der alte TSS-Segmentselektor als Rückverweis auf das alte Task-State-Segment eingetragen.
Die im Task-State-Segment abgespeicherten Stackpointerwerte (SS:SP) für die Privilegierungsstufen CPL=0, CPL=1 und CPL=2 sind unveränderlich. Sie werden vom Programmierer bei der Systeminitialisierung für den Fall vergeben, daß innerhalb einer laufenden Task die Privilegierungsstufe einzelner Routinen mit Hilfe eines CALL-Gateübergangs erstmals verändert wird (Abschnitt 7. Zugriffsschutz).
Erfolgt in einer Task keine Veränderung auf eine der drei Privilegierungsstufen 0, 1 oder 2, sind die entsprechenden Stackpointerinitialwerte im Task-State-Segment belanglos.
Die Anfangsposition des Stack nach einem Taskwechsel wird immer den Registern SS und SP entnommen, die auch Teil des Taskstatus im Task-State-Segment sind.
Zu beachten ist, daß beim Wechsel der Privilegierungsstufe über einen Gateübergang der Stackpointer der aufrufenden Routine in den neuen Stack der neu aufgerufenen Routine und nicht im Task-State-Segment abgelegt wird (Bild 7.2).

8.5. Taskwechsel

Ein Taskwechsel kann auf eine der folgenden vier Arten ausgelöst werden:

1. Die Selektorkomponente der Adresse eines langen (far) JMP- oder CALL-Befehls zeigt auf einen TSS-Deskriptor in der Globaldeskriptortabelle (GDT).
   Die Offsetkomponente der Adresse im langen JMP- oder CALL-Befehl wird in diesem Fall ignoriert - sie ist belanglos, muß aber vorhanden sein.

2. Ein IRET-Befehl wird zu einem Zeitpunkt ausgeführt, in dem das NT-Bit (NT nested task) im Flagregister den Wert 1 besitzt.
   Der auf den neuen TSS-Deskriptor zeigende TSS-Segmentselektor wird in diesem Fall aus dem Back-Link-Feld des momentan aktuellen Task-State-Segments entnommen.
   Der Taskwechsel über einen IRET-Befehl bei NT=1 führt immer zur Rückkehr zu einer Task, die über einen CALL-Befehl oder einen Interrupt verlassen wurde.
   Der Taskwechsel bei Taskverkettung wird im Abschnitt 8.6. (Taskverkettung) ausführlich behandelt.

3. Die Selektorkomponente der Adresse eines langen JMP- oder CALL-Befehls zeigt auf einen sogenannten Taskgatedeskriptor in der Globaldeskriptortabelle (GDT) oder in der Lokaldeskriptortabelle (LDT).
   In diesem Fall befindet sich der neue TSS-Segmentselektor in dem Taskgatedeskriptor.
   Die Offsetkomponente der Adresse im langen JMP- oder CALL-Befehl wird, wie oben, ignoriert.
   Der Taskwechsel über Taskgatedeskriptoren wird im Abschnitt 8.7. (Taskgatedeskriptoren) ausführlich behandelt.

4. Ein von der 80286-CPU anerkannter Interrupt zeigt mit seinem Interruptvektor auf einen Taskgatedeskriptor in der Interruptdeskriptortabelle (IDT). Der neue TSS-Segmentselektor befindet sich in dem Taskgatedeskriptor (Abschnitte 8.7. Taskgatedeskriptor und 9. Interruptsystem).

Wird der Taskwechsel über einen CALL-Befehl oder über einen Interrupt ausgelöst, impliziert dies, daß eine spätere Rückkehr zu der momentan ausgeführten Task möglich ist.
Bei der Abwicklung des Taskwechsels über einen JMP- oder IRET-Befehl ist das nicht der Fall.

Ein Taskwechsel selbst läuft immer in folgenden fünf Schritten ab:

1. Überprüfung, ob die aktuelle Privilegierungsstufe (CPL) des einen Taskwechsel auslösenden JMP- oder CALL-Befehls numerisch kleiner oder gleich der im TSS-Deskriptor festgelegten Deskriptorprivilegierungsstufe DPL des Task-State-Segments ist.

2. Überprüfung, ob das Presentbit (P) im TSS-Deskriptor der aufrufenden Task den Wert 1 besitzt (TSS-Deskriptor ist dann gültig) und ob die im TSS-Deskriptor eingetragene TSS-Länge mindestens den Wert 43 (002BH) besitzt.

Achtung! Werden im Schritt 1 und 2 Fehler ermittelt, wird die Behandlung dieser Fehler durch die dann nicht ausscheidende alte Task übernommen.

3. Rettung der CPU-Basisregister der ausscheidenden Task in den veränderlichen Teil des alten Task-State-Segments (DS, SS, CS, ES, DI, SI, BP, SP, BX, DX, CX, AX, Flags, IP).
   Der gerettete Befehlszähler (IP) zeigt auf den Befehl, der auf den den Taskwechsel auslösenden Befehl folgt.
   Das Busy-Bit (B) im Accessbyte des TSS-Deskriptors der ausscheidenden Task wird rückgesetzt (B=0).

4. Das Taskregister der 80286-CPU wird mit dem TSS-Segmentselektor der aufrufenden Task geladen. Das Busy-Bit (B) im Accessbyte des TSS-Deskriptors der aufgerufenen Task wird auf 1 gesetzt (TSS besetzt - Task läuft).

5. Aus dem Task-State-Segment der aufgerufenen Task werden die Werte der folgenden CPU-Register neu geladen: LDT-Segmentselektor, DS, SS, CS, ES, SI, BP, SP, BX, DX, CX, AX, Flags, IP.

Achtung! Werden bei diesem Laden der CPU-Register Plausibilitätsfehler ermittelt, erfolgt die Fehlerbehandlung in der aufgerufenen Task.

Jeder Taskwechsel führt außerdem grundsätzlich zum Setzen des Taskswitch-bit (TS) im Maschinenstatuswort des Mikroprozessors 80286. Das TS-Bit wird besonders im Zusammenhang mit dem Numerikcoprozessor 80287 genutzt (Abschnitt 4.5. Maschinenstatuswort).
In der Tafel 8.1 sind alle bei einem Taskwechsel automatisch ausgeführten Fehlerüberprüfungen und Flag-Bit-Beeinflussungen zusammengefaßt.

Tafel 8.1. Fehlerüberprüfungen und Flag-Bit-Beeinflussungen bei einem Taskwechsel

Fehlerüberprüfungen

Testschritt	Fehlertyp	Fehlerkode
1. Neuer TSS-Deskriptor P=1	NP (not present)	TSS-Selektor
2. Neuer TSS-Deskriptor B=0	GP (general protection)	TSS-Selektor
3. TSS-Länge 43	TSS fehlerhaft (invalid)	TSS-Selektor
Umladen aller Register!		
4. LDT gültig	TSS fehlerhaft (invalid)	TSS-Selektor
5. LDT vorhanden (P=1)	TSS fehlerhaft (invalid)	TSS-Selektor
6. CS-Selektor gültig	TSS fehlerhaft (invalid)	CS-Selektor
7. Kodesegment vorhanden (P=1)	NP (not present)	CS-Selektor
8. Kodesegment DPL/RPL-Test	TSS fehlerhaft (invalid)	CS-Selektor
9. Stacksegment gültig	SF (stack fault)	SS-Selektor
10. Stacksegment plausibel	GP (general protection)	SS-Selektor
11. Stacksegment vorhanden	SF (stack fault)	SS-Selektor
12. Stacksegment DPL/CPL-Test	SF (stack fault)	SS-Selektor
13. DS/ES-Selektor gültig	GP (general protection)	DS/ES-Selektor
14. DS/ES-Segment lesbar	GP (general protection)	DS/ES-Selektor
15. DS/ES-Segment vorhanden	NP (not present)	DS/ES-Selektor
16. DS/ES-Segment DPL/CPL-Test	GP (general protection)	DS/ES-Selektor

Flag-Bit- und Back-Link-Beeinflussung beim Taskwechsel

	JMP	CALL / INT	IRET
B-Bit neuer TSS-Deskr.	B=1	B=1	unverändert
B-Bit alter TSS-Deskr.	B=0	unverändert	B=0
NT-Bit neues Flagreg.	NT=0	NT=1	unverändert
NT-Bit altes Flagreg.	unverändert	unverändert	NT=0
Back-Link neues TSS	unverändert	wird gesetzt	unverändert
Back-Link altes TSS	unverändert	unverändert	unverändert

8.6. Taskverkettung

Die Verkettung von zwei oder mehreren unterschiedlichen Tasks (task linking) baut auf dem schon behandelten Back-Link-Feld im Task-Status-Segment und auf dem NT-Bit (nested task) im Flagregister des Mikroprozessors 80286 auf.
Die Taskverkettung wird bei einem Taskwechsel mit einem CALL-Befehl und beim Taskwechsel über einen Interrupt aktiv.
Grundsätzlich gilt, daß bei jeder Art von Taskwechsel über einen CALL-Befehl oder einen Interrupt in das Back-Link-Feld der aufgerufenen Task der TSS-Segmentselektor der ausscheidenden Task eingetragen wird.
Wird ein Taskwechsel über einen CALL-Befehl oder einen Interrupt initiiert, wird außerdem das NT-Bit im Flagregister der eintretenden Task auf 1 gesetzt.
Damit wird angezeigt, daß eine verschachtelte Taskstruktur existiert, bei der das Back-Link-Feld die dann besonders wichtige Information beinhaltet, aus welcher Task heraus der Taskwechsel zur momentan laufenden Task ausgeführt wurde.
Die Rückkehr zur aufrufenden Task erfolgt in beiden Fällen - also sowohl bei einem vorherigen CALL-Taskwechsel als auch bei einem vorherigen interruptgesteuerten Taskwechsel - mit Hilfe eines IRET-Befehls.
Dieser Befehl wertet in jedem Fall den Zustand des NT-Bits aus. Hat dieses Bit im Flagregister den Wert 1, wird vom IRET-Befehl ein kompletter Taskwechsel ausgelöst, bei dem der neue TSS-Segmentselektor dem Back-Link-Feld der momentan laufenden Task entnommen wird. Dadurch erfolgt hardwaregesteuert die Rückkehr zur vorher laufenden Task.
Der gesamte Prozeß des Setzens (=1), aber auch des Rücksetzens (=0) des NT-Bits im aktuellen Flagregister der Task erfolgt, ebenso wie beim Busy-Bit (B) im Accessbyte des TSS-Segmentdeskriptors, automatisch ohne Mitwirkung des Programmierers.
Wird zu einem Zeitpunkt, in dem das NT-Bit im Flagregister 80286 den Wert 0 hat, ein IRET-Befehl ausgeführt, hat das zur Folge, daß eine ganz 'normale' Interrupt-Return-Operation abläuft, ohne daß damit ein Taskwechsel einhergeht.
Die Verschachtelungstiefe von Taskwechseloperationen ist im Rahmen der allgemein beim Mikroprozessor 80286 vorhandenen Grenzen unbeschränkt.

8.7. Taskgatedeskriptoren

Taskgatedeskriptoren unterstützen den indirekt aktivierten Taskwechsel beim Mikroprozessor 80286 im PVA-Mode.
Taskgatedeskriptoren werden wie 'normale' Gatedeskriptoren von JMP- und CALL-Befehlen, insbesondere aber von Interruptanforderungen genutzt (Abschnitt 7. Zugriffsschutz).
Sie dienen der Auslösung eines Taskwechsels, bei dem der TSS-Segmentselektor nicht dem Befehl selbst, sondern dem Taskgatedeskriptor entnommen wird. Ein Taskwechsel über einen Taskgatedeskriptor kann mit einer indirekten Adressierungsmöglichkeit verglichen werden.
Taskgatedeskriptoren können sich sowohl in der Globaldeskriptortabelle (GDT) als auch in der Lokaldeskriptortabelle (LDT) und der Interruptdeskriptortabelle (IDT) befinden, während sich ein TSS-Deskriptor ausschließlich in der Globaldeskriptortabelle (GDT) befinden darf.

Die Bedeutung von Taskgatedeskriptoren liegt vor allem in der außerordentlichen Flexibilität, die mit ihrer Hilfe beim Zugriff von 'lokalen Programmen' auf 'globale Systemdienste' erreicht werden kann.
Bei einem durch eine Interruptanforderung ausgelösten Taskwechsel erfolgt der Übergang in die neue Task immer über einen Taskgatedeskriptor in der Interruptdeskriptortabelle (IDT) (Abschn. 9. Interruptsystem).
Die Auswertung des Informationsgehalts der Felder P (Present-Bit) und DPL (Privilegierungsstufe) erfolgt bei Taskgatedeskriptoren in der in den Abschnitten 7. (Zugriffsschutz) und 8.2. (TSS-Deskriptor) beschriebenen Weise.
Ein Taskgatedeskriptor enthält in insgesamt acht Bytes folgende Parameter (Abschn. 6.2.2. Systemdeskriptoren; siehe auch Bild 6.2 und Tafel 6.4 bzw. 6.5):

	7							0	7							0	
+7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+6
+5	P	DPL		0	0	1	0	1	x	x	x	x	x	x	x	x	+4
+3	TSS-Segmentselektor													0	x	x	+2
+1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	+0

x nicht benutzt

8.8. Task-Status-Alias-Segmente

In Softwaresystemen, in denen die Notwendigkeit besteht, auf ein Task-Status-Segment direkt von einem Programm über Lese- oder Schreiboperationen zuzugreifen, ist es möglich, in der Globaldeskriptortabelle (GDT) mit Hilfe einfacher Datensegmentdeskriptoren einen sogenannten Task-Status-Alias-Segmentdeskriptor zu definieren, der ein 'normales' Task-Status-Segment mit einem 'normalen' TSS-Deskriptor überdeckt (Abschn. 6.2.4. Alias-Speichersegmente).
Interessanterweise muß das Alias-Segment nicht unbedingt die gleiche Länge wie das Task-Status-Segment besitzen. Dadurch ist es möglich, eine Task über die im Task-Status-Segment befindlichen Standardwerte hinaus mit zusätzlichen Parametern zu versorgen.

8.9. Tasksystemaktivierung (Urtask)

Die erste nach einem Systemstart (Reset) und dem Umschalten vom RA-Mode in den PVA-Mode aktivierte Task wird als Urtask in einem 80286-Mikroprozessorsystem bezeichnet.

Die Urtask sollte - obwohl auch andere Möglichkeiten existieren - immer mit Hilfe einer JMP-Taskwechseloperation initiiert werden.
Dabei ist zu beachten, daß sich vor der Ausführung dieser ersten JMP-Taskwechseloperation im Taskregister TR bereits ein gültiger TSS-Segmentselektor befinden muß, der über einen TSS-Segmentdeskriptor in der Globaldeskriptortabelle (GDT) auf ein Task-Status-Segment verweist. Dieses 'Dummy-TSS' dient zur Aufnahme der eigentlich belanglosen alten Programmstatuswerte beim Start der Urtask.
Der TSS-Segmentselektor des 'Dummy-TSS' muß nach dem Umschalten vom RA-Mode in den PVA-Mode und vor der JMP-Taskwechseloperation zur Urtask mit dem LTR-Befehl (Laden Taskregister) in das Taskregister (TR) der 80286-CPU geladen werden.
Nach dem Start der Urtask kann der Inhalt des 'Dummy-TSS' verworfen werden. Der zugehörige Speicherbereich steht für andere Aufgaben zur Verfügung (Abschn. 14. Systeminitialisierung).
Die im Task-State-Segment abgelegten Registerinitialwerte der Urtask selbst müssen, wie bei jedem anderen JMP-Taskwechsel, gültig sein, soll der Urtask-Startversuch erfolgreich verlaufen.

9. Interruptsystem

Das Interruptsystem des Mikroprozessors 80286 unterscheidet durch externe und durch interne Ereignisse ausgelöste Unterbrechungsanforderungen. Beide dienen der schnellen Reaktion auf Alarme, Fehlermeldungen, Bedieneranforderungen, Fertigmeldungen, Zeitsignale, Statuszustände u.a.m.
Externe Interruptanforderungen (Hardwareinterruptsignale) werden an den Mikroprozessor 80286 über die beiden speziell diesem Zweck vorbehaltenen Signalanschlußpins INTR und NMI übergeben.
Interne Interruptanforderungen (Softwareinterruptsignale) werden bei der Abarbeitung des INT-Befehls und beim Auftreten einer Ausnahmebedingung generiert.
Ausnahmebedingungen, auch Exceptions genannt, treten auf, wenn ein Befehl nicht ordnungsgemäß abgearbeitet werden kann (z.B. Division durch null, Stacküberlauf im PVA-Mode, Verletzung der Privilegierungsstufe u.a.m.).
Der Unterschied zwischen externen und internen Interrupts besteht im wesentlichen darin, daß externe Interruptsignale vollkommen asynchron, also unabhängig von der Ausführung des laufenden Programms, auftreten, während interne Interrupts ihre Ursache in der Abarbeitung von speziellen Befehlen im momentan laufenden Programm haben.
Bei der Behandlung der Interruptarchitektur des Mikroprozessors 80286 sind die beiden grundsätzlich unterschiedlichen Betriebsarten dieses Schaltkreises - REAL ADDRESS MODE und PROTECTED VIRTUAL ADDRESS MODE - zu berücksichtigen.

9.1. Externe Interruptanforderungen (INTR/NMI)

Externe Interruptanforderungen werden über die beiden speziell dieser Aufgabe zugeordneten Signalanschlußpins INTR und NMI an den Mikroprozessor 80286 übergeben.
Beide Interruptsignaleingänge unterscheiden sich dadurch voneinander, daß der INTR-Eingang durch Softwaremaßnahmen vom Programm aus maskiert - also abgeschaltet - werden kann.
Diese Maskierung des INTR-Eingangs erfolgt durch Löschen des Interruptflags IF (interrupt enable flag) im Flagregister (IF=0).
Der NMI-Eingang ist nicht maskierbar. Er sollte einer einzelnen höchstpriorisierten Interruptquelle vorbehalten bleiben (Netzausfall od.ä.).
Alle durch den Anwender des 80286 frei nutzbaren Interruptquellen müssen den maskierbaren INTR-Eingang gemeinsam benutzen. Die Unterscheidung, um welches Interruptsignal es sich handelt, wird durch die Übergabe eines 8-Bit-Interruptvektors (Interruptidentifikationsnummer) im Interruptanerkennungszyklus von einem externen Interruptsteuerschaltkreis (interrupt controller) an die 80286-CPU erreicht.
Durch die Größe des 8-Bit-Interruptvektors wird die Anzahl der unterschiedlichen INTR-Interruptquellen auf maximal 256 begrenzt.
Von diesen 256 möglichen Interruptvektoren stehen dem Anwender des Mikroprozessors 80286 real 224 zur freien Nutzung zur Verfügung. Die restlichen 32 INTR-Interruptsignalquellen sind durch feste interne Zuordnungen in der 80286-CPU oder aus Kompatibilitätsforderungen heraus reserviert. Das gilt für beide mögliche Betriebsarten dieses Schaltkreises.

Dem nichtmaskierbaren NMI-Interrupt ist der Interruptvektor 2 fest zugeordnet. Ein Interruptanerkennungszyklus mit Interruptvektorübergabe wie bei einem maskierbaren INTR-Interrupt existiert hier nicht.
Werden mehrere NMI-Quellen dem NMI-Signalpin des 80286 zugeordnet, muß die Behandlung zunächst immer in einer einheitlichen Serviceroutine erfolgen.
Beim Eintreffen eines NMI-Interrupts wird die dem Interruptvektor 2 zugeordnete NMI-Serviceroutine direkt aufgerufen. Solange diese Serviceroutine läuft, wird keine erneute NMI-Programmunterbrechung akzeptiert. Erst ein abschließender IRET-Befehl gibt diese interne Sperre wieder frei. Eine (und nur eine) zwischenzeitlich eingegangene erneute NMI-Anforderung wird allerdings automatisch gespeichert - geht also nicht verloren.
Für den Programmierer ist wichtig, zu beachten, daß in einer NMI-Serviceroutine das IF-Flag (interrupt enable flag) im Flagregister rückgesetzt werden muß (IF=0), um zu verhindern, daß der NMI-Serviceprozeß durch einen INT-Interrupt unterbrochen wird.
Zwischen den beiden externen Interruptsignalen INTR und NMI existieren also keine automatischen Verriegelungsmechanismen.
Das Löschen des IF-Flags kann programmgesteuert oder im PVA-Mode auch durch Zuordnung eines Interruptgatedeskriptors zum Interruptvektor 2 erfolgen, da bei der Interruptbedienung über einen Interruptgatedeskriptor im PVA-Mode das Löschen des IF-Flag automatisch erfolgt (Abschn. 9.5. Interruptarchitektur PVA-Mode).

9.2. Interne Interruptanforderungen

Interne Interruptanforderungen haben ihre Ursache in der Abarbeitung eines INT-Befehls oder in der Abarbeitung eines Befehls, der eine Ausnahmebedingung (Exception) zur Folge hat.
Interne Interruptanforderungen können nicht maskiert werden.
Der INT-Befehl existiert in zwei Varianten:

INTn — Erzeugung eines internen Softwareinterrupts mit dem Interruptvektor n. Für n kann jede Zahl zwischen 0 und 255 stehen.

INT3 — Erzeugung eines internen Softwareinterrupts mit dem Interruptvektor 3 (Breakpoint-Interrupt).

In der ersten Variante kann der INT-Befehl zur Softwareerzeugung jedes beliebigen Interrupts, etwa für die Testung von Interruptserviceroutinen, für den Einsprung in Betriebssystemroutinen u.a.m., verwendet werden.
Der 1-Byte-Unterbrechungsbefehl INT3 dient der Auslösung eines Breakpoint-Interrupts für Testzwecke. Er wird insbesondere in Softwaredebuggern zur Anwendung kommen.
Alle internen Interruptanforderungen, die ihre Ursache in der Abarbeitung von Befehlen haben, die zu einer Ausnahmebedingung führen, werden als Exceptions bezeichnet.
Exceptions weisen immer auf Softwarefehler in Datenkonstellationen oder auf Zugriffsschutzverletzungen hin. Ihnen sind in der Folge genau zu behandelnde Interruptvektoren fest zugeordnet.

Befehle, bei deren Abarbeitung durch fehlerhafte Datenkonstellationen eine Ausnahmebedingung entstehen kann, sind der INTO-Befehl (overflow), der DIV-Befehl (Division durch null), der BOUND-Befehl (Verletzung von Feldgrenzen) u.a.

9.3. Reservierte Interruptvektoren

Das Interruptsystem des Mikroprozessors 80286 kennt in beiden möglichen Betriebsarten - RA-Mode und PVA-Mode - jeweils 32 reservierte Interruptvektoren (Tafel 9.1).

Tafel 9.1. Übersicht über die reservierten Interruptvektoren des Mikroprozessors 80286

Interruptvektor 0 Divisionsfehler-Exception Ursache im RA-Mode und PVA-Mode: Abarbeitung einer vorzeichenlosen Division (DIV-Befehl) oder einer Integerdivision (IDIV-Befehl) mit Divisor 0 oder Quotientenüberlauf. Die Rücksprungadresse im Stack zeigt auf das erste Byte des fehlerauslösenden Befehls (siehe auch Abschn. 15. Kompatibilität). Im PVA-Mode wird kein Fehlerkode von der 80286-CPU an den Interrupthandler im Stack übergeben.
Interruptvektor 1 Einzelschrittinterrupt (single step) Ursache im RA-Mode und PVA-Mode: Abarbeitung eines beliebigen Befehls bei gesetztem TF-Flag (TF=1) im Flagregister des 80286 (TF trap flag). Bei der Interruptanerkennung wird der alte Flagregisterinhalt in den Stack abgespeichert und das TF-Flag gelöscht (TF=0). In der behandelnden Interruptserviceroutine ist die Einzelschrittabarbeitung dann bis zum automatischen Rückspeichern des alten Flagregisterinhalts (TF=1) mit dem IRET-Befehl ausgeschaltet. Der Einzelschrittinterrupt wird von Softwaredebuggern, insbesondere zur Realisierung von Tracefunktionen, genutzt. Die Rücksprungadresse im Stack zeigt auf das erste Byte des nächsten Befehls. Im PVA-Mode wird kein Fehlerkode von der 80286-CPU an den Interrupthandler im Stack übergeben.

Interruptvektor 2 NMI-Interrupt (no maskable interrupt)

Ursache im RA-Mode und PVA-Mode:

Eintreffen eines NMI-Interruptsignals am NMI-Signalpin der 80286-CPU.

Die Rücksprungadresse im Stack zeigt auf das erste Byte des nächsten Befehls.

Im PVA-Mode wird kein Fehlerkode von der 80286-CPU an den Interrupthandler im Stack übergeben.

Interruptvektor 3 Breakpoint-Interrupt (breakpoint interrupt)

Ursache im RA-Mode und PVA-Mode:

Abarbeitung eines INT3-Befehls. Der INT3-Befehl wird als 1-Byte-Breakpoint-Befehl (Operationskode: OCCH) zur Erzeugung eines Haltepunktes in Softwaredebuggern verwendet.

Die Rücksprungadresse im Stack zeigt auf das erste Byte der auf den INT3-Befehl folgenden Befehlsfolge.

Achtung! Wird der INT3-Befehl zur Erzeugung eines Haltepunktes in einem Programm anstelle eines anderen Befehls eingesetzt, muß nach der Rückspeicherung des alten Befehlskodes der IP-Registerinhalt im Stack in der Breakpoint-Serviceroutine um 1 dekrementiert werden.

Im PVA-Mode wird kein Fehlerkode von der 80286-CPU an den Interrupthandler im Stack übergeben.

Interruptvektor 4 Überlauf-Exception (overflow exception)

Ursache im RA-Mode und PVA-Mode:

Ist bei der Abarbeitung eines INTO-Befehls (interrupt if overflow) das OF-Bit im Flagregister der 80286-CPU gesetzt, erfolgt die automatische Generierung dieses Interrupts.
Der INTO-Befehl dient der Erzeugung eines Softwareinterrupts bei einem über das OF-Flag erkannten Datenüberlauf nach arithmetischen Operationen.

Die Rücksprungadresse im Stack zeigt auf das erste Byte des nächsten Befehls.

Im PVA-Mode wird kein Fehlerkode von der 80286-CPU an den Interrupthandler im Stack übergeben.

Interruptvektor 5 BOUND-Exception

Ursache im RA-Mode und PVA-Mode:

Bei der Abarbeitung eines BOUND-Befehls wird ein Wert außerhalb der spezifizierten Feldgrenzen ermittelt. Es handelt sich dabei immer um einen Softwarefehler (siehe auch Abschn. 13. Befehlssatz).

Die Rücksprungadresse im Stack zeigt auf das erste Byte des fehlerhaften Befehls.

Im PVA-Mode wird kein Fehlerkode von der 80286-CPU an den Interrupthandler im Stack übergeben.

Interruptvektor 6 Operationskode-Exception (invalid opcode exeption)

Ursache im RA-Mode und PVA-Mode: Abkürzung: #UD

Bei der Abarbeitung eines Programms stößt der Mikroprozessor 80286 auf einen fehlerhaften, nicht definierten Operationskode oder einen Operationskode mit falschen Attributen.
Dieser Interrupt zeigt fehlerhafte Programmkodesequenzen an, kann aber auch verwendet werden, um Erweiterungen des 80286-Befehlssatzes in einer zugehörigen Interruptserviceroutine zu implementieren.
Ein fehlerhafter Operationskode kann z.B. sein:

- falscher oder nicht existierender Befehlskode im ersten oder zweiten Byte des Befehls,
- falsch definierter Operand (Register, Speicheradresse usw.), der nicht in Verbindung mit dem Befehlskode verwendet werden darf.

Die Rücksprungadresse im Stack zeigt auf das erste Byte des fehlerhaften Befehls.

Im PVA-Mode wird kein Fehlerkode von der 80286-CPU an den Interrupthandler im Stack übergeben.

Interruptvektor 7 Coprozessor-Extension-Exception

Ursache im RA-Mode und PVA-Mode: Abkürzung: #NM

Bei der Abarbeitung eines ESC-Befehls ist das EM-Bit (emulate flag) im Maschinenstatuswort (MSW) der 80286-CPU gesetzt, oder der Mikroprozessor trifft auf einen WAIT-Befehl, und sowohl das MP-Bit (math present) als auch das TS-Bit (task switched) im Maschinenstatuswort sind gesetzt.

Der erste Fall zeigt immer eine in der Interruptserviceroutine zu emulierende, extern nicht vorhandene Coprozessorfunktion an. Im zweiten Fall muß die 80286-CPU auf die Fertigmeldung einer Coprozessoroperation nach einem Taskwechsel warten.

Die Rücksprungadresse im Stack zeigt auf das erste Byte des die Exception auslösenden Befehls.

Im PVA-Mode wird kein Fehlerkode von der 80286-CPU an den Interrupthandler im Stack übergeben.

Interruptvektor 8 Doppelfehler-Exception

Ursache im RA-Mode:

Es tritt ein Interruptvektor außerhalb der definierten Größe der Startadressentabelle aller Interruptserviceroutinen auf.
Die Größe der Interruptserviceroutinen-Startadressentabelle im Speicher des 80286-Mikrorechners wurde mit einem LIDT-Befehl kleiner als 03FFH festgelegt.

Die Rücksprungadresse im Stack zeigt auf das erste Byte des Befehls, der die Exception auslöste.

Ursache im PVA-Mode: Abkürzung: #DF

Die Doppelfehler-Exception tritt auf, wenn bei der Behandlung eines Ausnahmebedingungsinterrupts (Exception) ein weiterer Ausnahmebedingungsinterrupt (Exception) wegen Stacküberlauf oder fehlerhaften bzw. nicht vorhandenen Segmenten (P=0 not present) auftritt.
Kann eine Doppelfehler-Exception wegen des Auftretens einer weiteren Ausnahmebedingung nicht behandelt werden (Dreifachfehler), schaltet sich die 80286-CPU automatisch ab (shut down state). Der Wiederanlauf ist danach nur über NMI oder RESET möglich.

Die Rücksprungadresse im Stack zeigt auf das erste Byte des Befehls, der die Exception auslöste.

Im PVA-Mode wird der Fehlerkode 0000H von der 80286-CPU an den Interrupthandler im Stack übergeben.

Interruptvektor 9 Coprozessor-Segment-Overrun-Exception

Ursache im RA-Mode und PVA-Mode: Abkürzung: #MP

Ein Speicheroperand, der von einem Coprozessor gelesen oder geschrieben wird, liegt außerhalb der Segmentgrenze.
Die Coprozessor-Segment-Overrun-Exception ist eine nichtmaskierbare asynchrone externe Interruptquelle, die ihre Ursache im z.B. in einem 80287-Numerikcoprozessor hat. Die Behandlung der Coprozessor-Segment-Overrun-Exception hängt vom verwendeten Coprozessor ab. Sie ist mit äußerster Sorgfalt vorzunehmen.

Die Rücksprungadresse im Stack zeigt auf einen Befehl im bearbeiteten 80286-Befehlsstrom, der in keiner Verbindung zu der Coprozessor-Segment-Overrun-Exception stehen muß.

Im PVA-Mode wird kein Fehlerkode von der 80286-CPU an den Interrupthandler im Stack übergeben.

Interruptvektor 10 TSS-Exception

Ursache im RA-Mode:

Der Interruptvektor 10 ist im RA-Mode reserviert (nicht nutzbar).

Ursache im PVA-Mode: Abkürzung: #TS(Selektor)

Die Task-State-Segment-Exception (TSS-Exception) wird von der 80286-CPU generiert, wenn einer der folgenden Fehler in einem Task-State-Segment (TSS) nach einem Taskwechsel entdeckt wird:

Fehler	Fehlerkode
TSS-Länge kleiner als 43	TSS-id und EXT-Bit
LDT-Selektor fehlerhaft oder LDT P=0	LDT-id und EXT-Bit
Stacksegmentselektor außerhalb des Limits	SS-id und EXT-Bit
Stacksegment kann nicht geschrieben werden	SS-id und EXT-Bit
Fehler Stacksegment DPL/CPL	SS-id und EXT-Bit
Stacksegmentselektor RPL ungleich CPL	SS-id und EXT-Bit
Kodesegmentselektor außerhalb des Limits	CS-id und EXT-Bit
Kodesegmentselektor kein Kodesegment	CS-id und EXT-Bit
Non-Conforming-Kodesegment DPL ungleich CPL	CS-id und EXT-Bit
Conforming Kodesegment DPL größer als CPL	CS-id und EXT-Bit
DS- oder ES-Selektor außerhalb des Limits	ES/DS-id und EXT-Bit
DS- oder ES-Segmente sind nicht lesbar	ES/DS-id und EXT-Bit

Die Behandlung der TSS-Exception muß in einer eigenständigen Task erfolgen, deren Task-State-Segment fehlerfrei ist.

Die Rücksprungadresse im Stack zeigt auf den nächsten auszuführenden Befehl.

Es wird ein Fehlerkode (id steht für Selektor), wie oben angegeben, von der 80286-CPU an den Interrupthandler im Stack übergeben.
Das EXT-Bit in der Bitposition 0 des 16-Bit-Fehlerkodes kennzeichnet bei der Auswertung:

EXT=1 Der Fehler ist nicht durch eine Aktion der laufenden Task entstanden (externe Interrupts, Single-Step-Interrupt, Coprozessorinterrupt usw.).

EXT=0 Der Fehler ist durch eine Aktion entstanden, die bei der Abarbeitung des Befehls eingeleitet wurde, dessen Adresse durch den im Stack befindlichen CS:IP-Wert gekennzeichnet ist.

Interruptvektor 11 Segment-not-present-Exception

Ursache im RA-Mode:

Der Interruptvektor 11 ist im RA-Mode reserviert (nicht nutzbar).

Ursache im PVA-Mode Abkürzung: #NP(Selektor)

Von der 80286-CPU wird auf einen Deskriptor (CS, DS, ES, TSS, LDT) zugegriffen, dessen P-Bit (P=0 not present) gelöscht ist und damit anzeigt, daß das Segment sich nicht im Primärspeicher befindet.
Dieser Interrupt wird bei virtuellen Adreßverwaltungssystemen verwendet, bei denen die zugehörige Interruptserviceroutine das Nachladen des Segments von einem externen Massendatenspeicher (Hard-Disk) übernimmt.

Die Rücksprungadresse im Stack zeigt auf das erste Byte des Befehls, der die Exception auslöste.

Es wird der gleiche Fehlerkode wie beim Interruptvektor 10 von der 80286-CPU an den Interrupthandler im Stack übergeben.

Interruptvektor 12 Stacksegment-Exception (stack fault)

Ursache im RA-Mode:

Der Interruptvektor 12 ist im RA-Mode reserviert (nicht nutzbar).

Ursache im PVA-Mode: Abkürzung: #SS(Selektor oder 0)

Von der 80286-CPU wird beim Laden des Stacksegmentdeskriptors auf einen Deskriptor zugegriffen, der in diesem Moment nicht vorhanden ist (P=0 not present), oder beim Einkellern bzw. beim Auskellern von Daten in den Stack tritt eine Stacksegmentbereichsverletzung auf (overflow/underflow).

Die Rücksprungadresse im Stack zeigt auf das erste Byte des Befehls, der die Exception auslöste.

Es wird der gleiche Fehlerkode wie beim Interruptvektor 10 von der 80286-CPU an den Interrupthandler im Stack übergeben.

Interruptvektor 13 General-Protection-Exception

Ursache im RA-Mode:

Die General-Protection-Exception tritt im RA-Mode auf, wenn auf ein Wort am Ende eines Segments zugegriffen wird, dessen Adresse mit OFFFFH beginnt, und damit eine Segmentüberschreitung vorliegt.

Die Rückkehradresse im Stack zeigt auf das erste Byte des Befehls, der die Exception auslöste.

Ursache im PVA-Mode: Abkürzung: #GP(Selektor oder 0)

Die General-Protection-Exception ist im PVA-Mode ein Sammelinterrupt, der eine allgemeine Privilegierungsschutzverletzung kennzeichnet, die ihre Ursache in einer Fülle von Möglichkeiten haben kann. Dazu zählen insbesondere

- Überschreitung der Segmentgrenzen von CS, DS, ES, TSS, LDT,
- Sprung in ein Datensegment,
- Taskwechsel zu einer bereits aktiven Task (busy),
- Schreiben eines Read-only- oder Execute-only-Segments,
- Lesen eines Execute-only-Segments,
- Priviligierungsschutzverletzungen (CPL, DPL, EPL, RPL),
- Zugriff über DS oder ES mit Nullselektor

u.a.m.

Die Rücksprungadresse im Stack zeigt auf das erste Byte des Befehls, der die Exception auslöste.

Es wird ein Fehlerkode von der 80286-CPU an den Interrupthandler im Stack übergeben.
Dieser Fehlerkode ist bei Segmentüberschreitungsfehlern 0000H.

Anderenfalls wird wie beim Interruptvektor 10 der Segmentselektor im Fehlerkodewort zurückgegeben.
Bei Fehlern, die ihre Ursache in Zugriffen auf die Interruptdeskriptortabelle (IDT) haben, wird in den Bitpositionen 10-3 des Fehlerkodeworts der verwendete Interruptvektor zurückgegeben. In diesem Fall wird außerdem das EXT-Bit, wie beim Interruptvektor 10 angegeben, gesetzt bzw. rückgesetzt.

Interruptvektoren 14 und 15 sind reserviert (nicht nutzbar).

Interruptvektor 16 Coprozessor-Exception

Ursache im RA-Mode und PVA-Mode: Abkürzung: #MF

Bei der Ausführung eines ESC- oder WAIT-Befehls wird, wenn das EM-Bit (emulate flag) im Maschinenstatuswort der 80286-CPU gesetzt ist (EM=1), das ERROR-Anschlußpin der 80286-CPU überprüft. Liegt hier ein aktives Signal an, wird die Coprozessor-Exception ausgelöst.
Da das ERROR-Anschlußpin des 80286 standardmäßig mit einem entsprechenden Signalpin des Coprozessors verbunden ist, erhält der Coprozessor die Möglichkeit, die CPU über verschiedene interne Fehler zu unterrichten.
Das kann bei einem Numerikcoprozessor 80287 beispielsweise sein:

- fehlerhafter Operationskode,
- Datenüberlauf,
- Division durch Null,
- nichtnormalisierter Operand,
- ungenaues Ergebnis.

Die Rücksprungadresse im Stack zeigt auf das erste Byte des Befehls, der die Exception auslöste.

Ein Fehlerkode kann in der Regel aus dem Coprozessor ausgelesen werden.

Interruptvektoren 17, 18, 19 bis 31 sind reserviert (nicht nutzbar).

9.4. Interruptarchitektur im RA-Mode

Im RA-Mode des Mikroprozessors 80286 gilt die im Bild 9.1 dargestellte relativ einfache Interruptstruktur.
Ein 8-Bit-Interruptvektor weist als Pointer in einer Interruptvektortabelle auf die Startadresse der zugehörigen Interruptserviceroutine.
Die Interruptvektortabelle mit den Startadressen aller Interruptserviceroutinen befindet sich auf festen Adressen (0000H bis 03FFH) im Speicher des 80286-Mikrorechners.
Eine Startadresse in der Interruptvektortabelle umfaßt jeweils einen 16-Bit-Segmentselektor und einen 16-Bit-Offsetanteil, zusammen 32 Bit.
Insgesamt können in der Interruptvektortabelle bis zu 256 Startadressen von Interruptserviceroutinen aufgenommen werden, die dann den maximal 256 möglichen Interruptquellen zugeordnet sind.

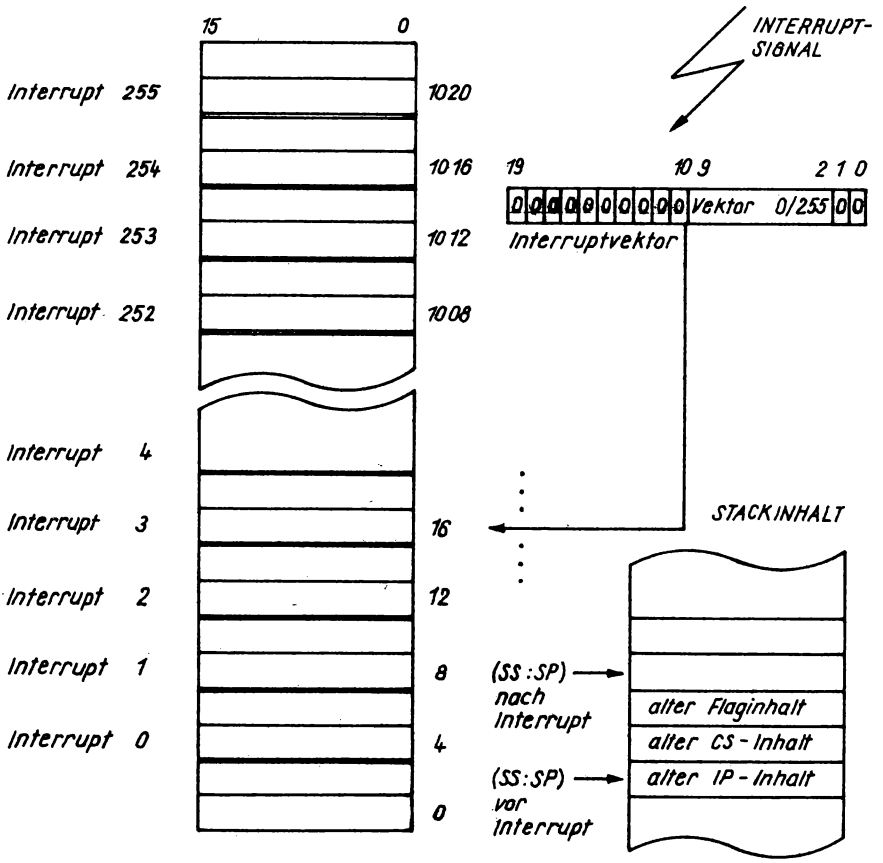


Bild 9.1. Interruptstruktur des Mikroprozessors 80286 im RA-Mode

Die ersten 32 Interruptvektoren sind im RA-Mode, genauso wie im noch zu behandelnden PVA-Mode, für feste Interruptereignisse reserviert - können vom Anwender also nicht für andere Zwecke genutzt werden (Abschn. 9.3. Reservierte Interruptvektoren).
Treffen mehrere Interruptsignale gleichzeitig ein, gilt die folgende Prioritätsfolge für deren Anerkennung:

1. Exception (Ausnahmebedingungen bei der Befehlsabarbeitung),
2. Einzelschrittinterrupt (single step interrupt),
3. NMI-Interrupt (non maskable interrupt),
4. Coprozessorfehlerinterrupt (coprocessor segment overrun),
5. INTR-Interrupt (maskable interrupt).

Diese Prioritätsreihenfolge gilt nur für die Interruptanerkennung bei gleichzeitigem Anliegen mehrerer Interruptsignale.
Die Interruptbearbeitung im RA-Mode des Mikroprozessors 80286 erfolgt nach dem Basisschema:

1. Abspeichern der momentanen (alten) Werte des CS-Registers, des IP-Registers und des Flagregisters in den Stack.
2. Löschen des IF- und TF-Flagbits im Flagregister der 80286-CPU:
   - IF=0 Ausschalten des maskierbaren Interruptsystems (interrupt enable flag),
   - TF=0 Ausschalten des Einzelschrittinterrupts (single step interrupt - trap flag).
3. Start der Interruptserviceroutine.
4. Rückkehr in das unterbrochene Programm mit dem IRET-Befehl.

Bei der Bearbeitung verschachtelter Interruptstrukturen gelten die grundsätzlich bei Mikroprozessorsystemen zu beachtenden Bedingungen für die zwischenzeitlich in einer Interruptserviceroutine mögliche Freigabe des maskierbaren Interruptsystems über das Setzen (IF=1) und Rücksetzen (IF=0) des Interruptflags im Flagregister der 80286-CPU.

9.5. Interruptarchitektur im PVA-Mode

Im PVA-Mode des Mikroprozessors 80286 existiert eine wesentlich komplexere Interruptarchitektur als im RA-Mode dieses Schaltkreises.
Sie unterstützt insbesondere die in den Abschnitten 7. (Zugriffsschutz) und 8. (Taskwechselmechanismen) behandelten Zugriffsschutz- und Taskwechselmechanismen.

9.5.1. Interruptdeskriptortabelle IDT

Der Mikroprozessor 80286 kann im PVA-Mode insgesamt 256 verschiedene Interruptquellen über einen Interruptvektor voneinander unterscheiden.
Für jeden dieser Interruptvektoren existiert in einer zusammenfassenden Interruptdeskriptortabelle (IDT) ein 8-Byte-Eintrag in Form eines Gatedeskriptors. Er stellt einen Verweis auf ein zu einem Interruptvektor gehörendes Interruptbehandlungsprogramm (Interruptserviceroutine) dar.

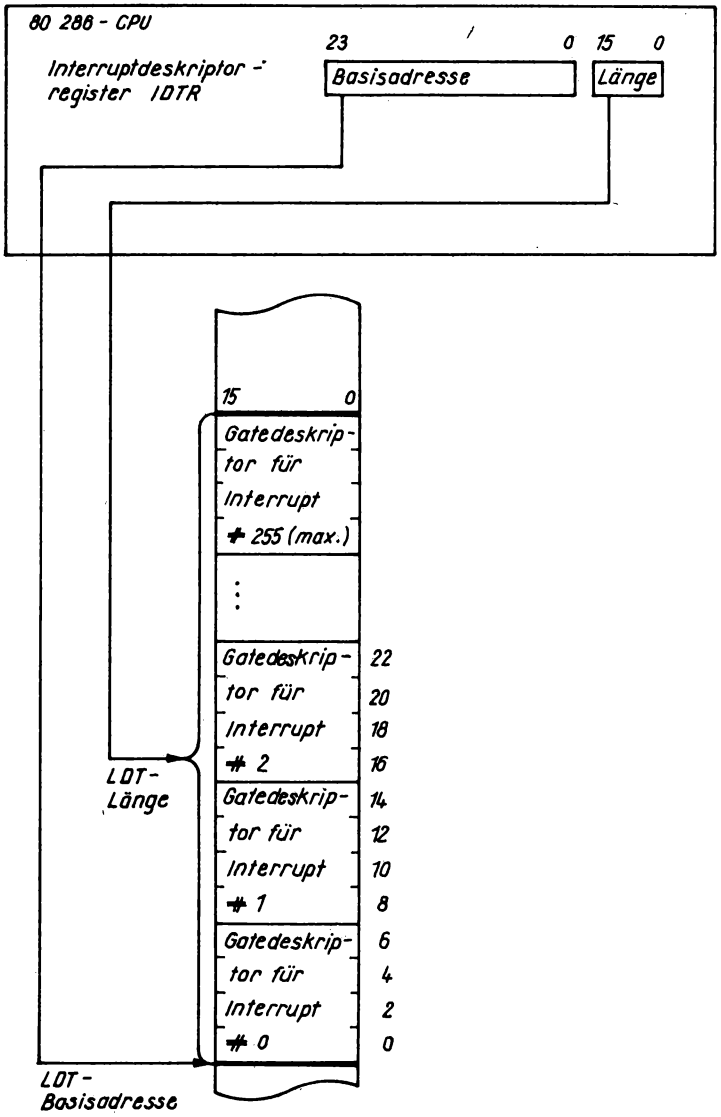


Bild 9.2. Aufbau der Interruptdeskriptortabelle beim Mikroprozessor 80286 im PVA-Mode

Die Anordnung der Interruptdeskriptortabelle (IDT) im Speicher eines 80286-Mikroprozessorsystems wird durch das in der 80286-CPU befindliche IDT-Register (IDTR) festgelegt. Es kann über einen speziellen Befehl (LIDT) bei der Systeminitialisierung mit einer 24-Bit-Basisadresse und einer 16-Bit-Länge geladen werden (Bild 9.2).

In der Interruptdeskriptortabelle (IDT) können drei verschiedene Gatedeskriptortypen vorkommen:

- Interruptgatedeskriptoren,
- Trapgatedeskriptoren und
- Taskgatedeskriptoren.

Wird von einem externen oder internen Interrupt auf einen Interruptgatedeskriptor oder einen Trapgatedeskriptor Bezug genommen, wird die Interruptbehandlung innerhalb der momentan ausgeführten Task vorgenommen.
Wird von einem externen oder internen Interrupt auf einen Taskgatedeskriptor Bezug genommen, hat das automatisch einen Taskwechsel zur Folge. Die Interruptserviceroutine wird dann in einer anderen als die zum Zeitpunkt der Interruptanerkennung laufenden Task behandelt.
Befinden sich in der Interruptdeskriptortabelle andere Deskriptoren als die genannten und wird auf einen solchen nicht zulässigen Deskriptor zugegriffen, hat das einen internen Fehlerinterrupt (GP general protection) zur Folge.
Es ist nicht erforderlich, daß alle 256 möglichen Einträge in einer Interruptdeskriptortabelle (IDT) in einem speziellen 80286-Softwaresystem auch genutzt werden.
Über die 16-Bit-Längenangabe im IDT-Register kann bei Bedarf eine verkürzte Interruptdeskriptortabelle definiert werden.
Befinden sich in der Interruptdeskriptortabelle ungenutzte Deskriptorpositionen, werden diese durch ein komplett auf Null gesetztes Accessbyte gekennzeichnet.
Erfolgt über einen externen oder internen Interrupt ein Zugriff auf einen Deskriptor außerhalb der durch die 16-Bit-Länge definierten Größe der Interruptdeskriptortabelle oder auf einen Deskriptor, dessen Accessbyte den Wert 00H besitzt, erfolgt, wie beim Zugriff auf fehlerhafte Deskriptoren in der Interruptdeskriptortabelle, ein interner Fehlerinterrupt (GP general protection).
Zu beachten ist, daß die Interruptvektoren 0 bis 31 für ganz fest definierte Interruptquellen reserviert sind. Die zugehörigen Gatedeskriptoren in der Interruptdeskriptortabelle bleiben deshalb immer diesen 32 vordefinierten Interruptquellen vorbehalten.
Die restlichen 224 Interruptquellen sind für den 80286-Anwender frei verfügbar.

9.5.2. Gatedeskriptoren Interruptdeskriptortabelle

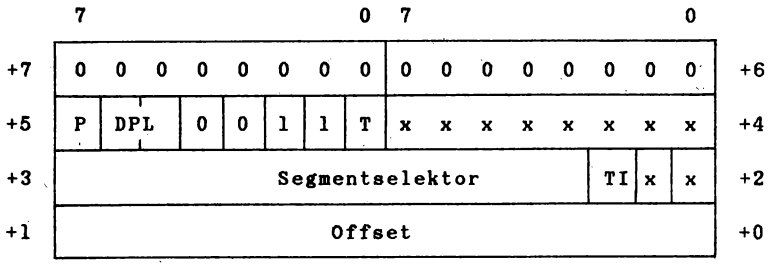
Wie schon erwähnt, können in der Interruptdeskriptortabelle drei grundverschiedene Gatedeskriptortypen Aufnahme finden:

- Interruptgatedeskriptoren,
- Trapgatedeskriptoren und
- Taskgatedeskriptoren.

Nur Interrupts, deren Behandlung über Taskgatedeskriptoren eingeleitet wird, führen in der Folge zu einem Taskwechsel, bei dem die Interruptserviceroutine als eigenständige Task läuft. In den beiden anderen Fällen erfolgt die Interruptbehandlung in der momentan bearbeiteten Task.

INTERRUPT-/TRAPGATEDESKRIPTOREN

Interrupt- und Trapgatedeskriptoren enthalten in insgesamt acht Bytes folgende Parameter (Abschn. 6.2.2. Systemdeskriptoren; siehe auch Bild 6.2 und Tafel 6.4 bzw. 6.5):



TI Auswahl GDT (TI=0) oder LDT (TI=1)

T T=1 Trapgatedeskriptor
T=0 Interruptgatedeskriptor

x nicht benutzt

Bei Interrupt- und Trapgatedeskriptoren wird, sehr ähnlich wie bei CALL-Gatedeskriptoren, ein Segmentselektor eines Programmkodesegments in der Lokal- oder Globaldeskriptortabelle und ein Eintrittspunkt (offset) in diesem Segment durch den Deskriptor vorgegeben.
Interrupt- und Trapgatedeskriptoren dürfen nur in der Interruptdeskriptortabelle (IDT) auftauchen. Beide dienen der Zuordnung eines in einem bestimmten Programmkodesegment der momentan laufenden Task befindlichen Interruptbehandlungsprogramms zu einem externen oder internen Interrupt. Ein Taskwechsel wird durch keinen dieser beiden Deskriptortypen ausgelöst. Der Unterschied zwischen Interrupt- und Trapgatedeskriptor liegt in der Behandlung des IF-Flag (interrupt enable flag) im Flagregister des 80286. Während beim Aufruf einer Interruptserviceroutine über einen Interruptgatedeskriptor das IF-Flag automatisch gelöscht wird (IF=0), bleibt es beim Aufruf über einen Trapgatedeskriptor unverändert.
Ein Aufruf einer Interruptserviceroutine über einen Interruptgatedeskriptor hat also zur Folge, daß weitere Interruptanforderungen zunächst erst einmal so lange nicht bedient werden, bis über das laufende Programm durch manuelles Setzen des IF-Flag (IF=1) weitere Interrupts wieder zugelassen werden.
Beim Aufruf über einen Trapgatedeskriptor erfolgt keine Statusveränderung des 80286-Interruptsystems.
Aus der eben beschriebenen Eigenschaft heraus werden Interruptdeskriptoren insbesondere bei der Behandlung externer Interrupts verwendet, wogegen Trapgatedeskriptoren bei der Behandlung von internen Exceptions Anwendung finden sollten.

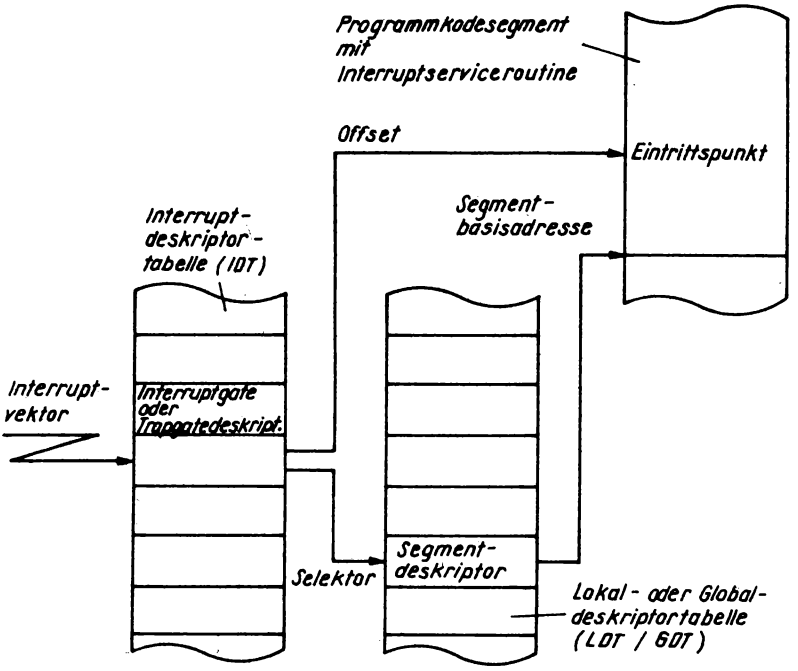


Bild 9.3. Interruptbehandlung mit Interrupt- und Trapgatedeskriptoren beim Mikroprozessor 80286

Die Behandlung des NT-Flags (nested task) im Flagregister des 80286 ist sowohl bei der Interruptbehandlung über einen Interruptgatedeskriptor als auch bei der über einen Trapgatedeskriptor einheitlich. In beiden Fällen wird dieses Bit im Flagregister, nach Abspeicherung des alten Flagregisterinhalts in den Stack, gelöscht (NT=0).
Das hat, wie schon im Abschnitt 8.6. (Taskverkettung) beschrieben, zur Folge, daß ein die Interruptserviceroutine abschließender IRET-Befehl bei Interrupt- und Trapgatedeskriptoren ohne einen Taskwechsel einhergeht.
Im Accessbyte eines Interrupt- oder Trapgatedeskriptors befindet sich, neben dem in der üblichen Weise benutzten P-Bit und dem einen Interrupt- oder Trapegatedeskriptor spezifizierenden Bitmuster '0011T', insbesondere die Deskriptorprivilegierungsstufe DPL (descriptor privilege level).
Dieses DPL-Feld wird bei allen in der Interruptdeskriptortabelle (IDT) befindlichen Deskriptortypen dazu benutzt, um eine momentane Privilegierungsstufe (CPL) zu definieren, die mindestens bei einem Programm vorhanden sein muß, das über einen INTn- oder INT3-Befehl den jeweiligen Interrupt-, Trap- oder Taskgatedeskriptor benutzen möchte.
Durch diesen Schutzmechanismus ist es möglich, über die im DPL-Feld in der IDT eingetragenen Werte eine bestimmte Privilegierungsstufe vorzuschrei-

ben, um nichtautorisierten Anwenderprogrammen den Weg zu privilegierten Systemdiensten über INT-Befehle zu versperren.
Bei externen Interruptquellen und bei Exceptions wird das DPL-Feld im Accessbyte des Interrupt- oder Trapgatedeskriptors ignoriert.
Im Zusammenhang mit Zugriffsschutzmechanismen sind sogenannte Conforming-Programmkodesegmente zu beachten. Sie erlauben, eine Interruptserviceroutine genau in der gleichen momentanen Privilegierungsstufe CPL abzuarbeiten, die das durch den externen oder internen Interrupt unterbrochene Programm gerade hatte.
Dies kann in einer Reihe von Fällen wünschenswert sein. Die Interruptserviceroutine wird in diesem Fall in einem Conformingsegment angeordnet. Die 80286-CPU setzt dann die momentane Privilegierungsstufe des laufenden Programms automatisch als Deskriptorprivilegierungsstufe DPL der Interruptserviceroutine ein - vorausgesetzt, der DPL-Wert in dem zugehörigen Conforming-Programmkodesegment ist kleiner oder gleich der geforderten momentanen Privilegierungsstufe CPL der unterbrochenen Prozedur.

TASKGATEDESKRIPTOREN

Das Interruptsystem des Mikroprozessors 80286 ermöglicht im PVA-Mode den direkten, über ein externes oder internes Interruptsignal ausgelösten Taskwechsel. Voraussetzung dafür ist, daß in der Interruptdeskriptortabelle den Interruptsignalen, die einen Taskwechsel auslösen sollen, ein Taskgatedeskriptor zugeordnet wird.
Die Taskgatedeskriptoren in der Interruptdeskriptortabelle (IDT) haben genau den gleichen Aufbau wie die Taskgatedeskriptoren in der Globaldeskriptortabelle (GDT) (Abschn. 8.7. Taskgatedeskriptoren).
Wird ein Taskgatedeskriptor verwendet, um Ausnahmebedingungsinterrupts (Exceptions) über die Interruptdeskriptortabelle zu einem Taskwechsel zu führen, wird der dann automatisch generierte Fehlerkode in den Stack der die Exceptionbehandlung durchführenden neuen Task eingespeichert, also nicht in den Stack der Task, die durch den Ausnahmebedingungsinterrupt unterbrochen wurde.
Die Behandlung von internen und externen Interruptsignalen in eigenständigen Tasks hat zwei wesentliche Vorteile:

1. Ein Taskwechsel ist immer mit einer automatischen Rettung und Umschaltung aller wesentlichen 80286-Register verbunden. Es werden bei einem Taskgateübergang nach einem Interrupt also nicht (wie beim Übergang über Interrupt- und Trapgatedeskriptoren) nur die Flagregister und der CS:IP-Wert gesichert.

2. Die durch den Interrupt initialisierte Task ist durch die beim Mikroprozessor 80286 verfügbaren Taskwechselmechanismen komplett von allen anderen Task abgeschirmt. Das gilt nicht nur für Registerwerte, sondern auch bezüglich der verfügbaren Adreßräume und der Zugriffsberechtigungen.
   Außerdem ist die Privilegierungsstufe der einen externen oder internen Interrupt behandelnden Task unabhängig von der Privilegierungsstufe der unterbrochenen Task.

Die Abwicklung des Taskwechsels erfolgt bei einem interruptgesteuerten Taskwechsel über ein Task-State-Segment (TSS) nach dem gleichen Verfahren

und mit den gleichen Randbedingungen wie bei einem Taskwechsel, der über einen CALL- oder JUMP-Befehl ausgelöst wurde.
Eine genaue Beschreibung dazu ist im Abschnitt 8. (Taskkonzept PVA-Mode) zu finden.
Besonders hervorgehoben werden sollte aber an dieser Stelle die genaue Behandlung des Busy-Bits (B) im TSS-Deskriptor (Besetztkennzeichen), des NT-Bits im Flagwort (nested task) und des Back-Link-Feldes im Task-State-Segment beim Eintritt (Interrupt) und beim Austritt (IRET) aus der Interruptserviceroutine, die als eigenständige Task läuft.

Eintrittstaskwechsel Interruptserviceroutine (Interrupt)		
	Unterbrochene Task	Interruptserviceroutinentask
B-Bit (busy)	unverändert	muß vorher 0 sein - wird auf 1 gesetzt
NT-Bit (nested task)	unverändert	wird auf 1 gesetzt
Back-Link-Feld	unverändert	Selektor zum TSS der unterbrochenen Task

Austrittstaskwechsel Interruptserviceroutine (IRET)		
	Unterbrochene Task	Interruptserviceroutinentask
B-Bit (busy)	unverändert	wird auf 0 rückgesetzt
NT-Bit (nested task)	unverändert	wird auf 0 rückgesetzt
Back-Link-Feld	unverändert	unverändert

9.5.3. Stackbehandlung

Bei der Behandlung von Interruptsignalen im PVA-Mode nutzt die 80286-CPU - wie im RA-Mode - den Stackdatenbereich zur Aufnahme verschiedener Parameter (Bild 9.4).
Dabei ist zu unterscheiden, ob die Interruptserviceroutine nach dem Interrupt- bzw. Trapgateübergang in der gleichen Privilegierungsstufe abgearbeitet wird wie das unterbrochene Programm oder ob sich die Privilegierungsstufe bei der Interruptbehandlung verändert. Im letzteren Fall wird, wie im Abschnitt 8. (Taskkonzept) beschrieben, für jede Privi-

legierungsstufe ein eigener Stackdatenbereich verwendet.
In dem beim Wechsel der Privilegierungsstufe dann neuen Stack werden neben Rückkehradressen, Flagregisterbelegungen und ggf. einem Fehlerkode dabei immer auch noch die Adressen des alten Stacks abgelegt.
Die unterschiedlichen Stackdatenbereiche werden generell, wie schon behandelt, im Task-State-Segment (TSS) jeder Task vordefiniert.
Bei der Abarbeitung einer über einen Taskgatedeskriptor initiierten Interruptserviceroutine als eigenständige Task erfolgt, wie im Abschnitt 8. (Taskkonzept) beschrieben, grundsätzlich ein Wechsel der Stackdatenbereiche aller Privilegierungsebenen. Zu beachten ist hier insbesondere, daß der bei einigen Ausnahmebedingungsinterrupts (Exceptions) automatisch entstehende Fehlerkode in den Stack der neuen Interruptserviceroutinentask eingetragen wird.
Die Rückkehr aus einer Interruptserviceroutine erfolgt im PVA-Mode, wie auch sonst üblich, mit einem IRET-Befehl. Dabei ist allerdings zu beachten, daß ein nach einer Exception im Stack befindlicher Fehlerkode vor dem IRET-Rücksprung aus dem Stack zu entfernen ist, um eine ordnungsgemäße Rückkehr sicherzustellen.
Wurde die Interruptserviceroutine über einen Interruptgatedeskriptor eingeleitet, wird durch die IRET-Befehlsabarbeitung das gesperrte Interruptsystem (IF=0) wieder freigegeben (IF=1).

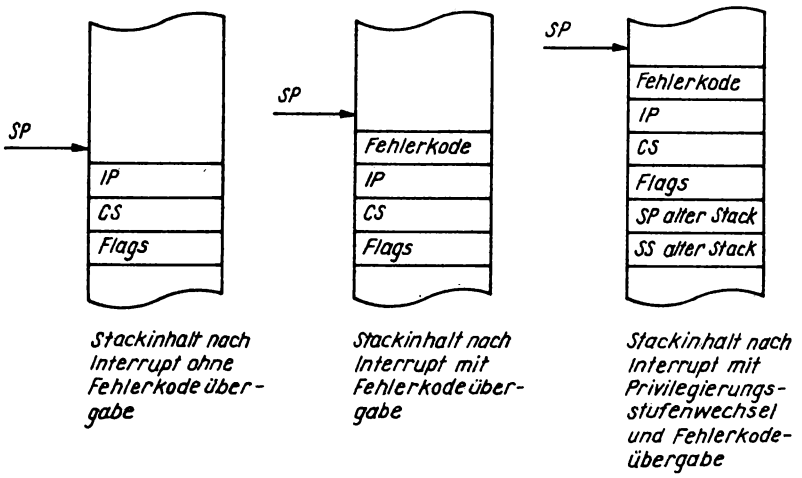


Bild 9.4. Stackinhalt bei der Interruptbehandlung über Interrupt- und Trapgatedeskriptoren beim Mikroprozessor 80286

9.5.4. Conforming-Interruptbehandlung

Im Zusammenhang mit der Interruptbehandlung über Interrupt- oder Trapgatedeskriptoren im PVA-Mode, die nicht zu einem Taskwechsel führen, spielen sogenannte Conforming-Programmkodesegmente eine besondere Rolle. Sie erlauben, eine Interruptserviceroutine genau in der gleichen momentanen Privilegierungsstufe (CPL) abzuarbeiten wie die, die das durch den externen oder internen Interrupt unterbrochene Programm gerade hatte.

Dies kann bei der Interruptbehandlung in verschiedenen Fällen wünschenswert sein. Die Interruptserviceroutine wird in diesem Fall in einem Conformingsegment angeordnet (Abschn. 7. Zugriffsschutz). Die 80286-CPU setzt dann die momentane Privilegierungsstufe (CPL) der Interruptserviceroutine ein - vorausgesetzt, der DPL-Wert in dem zugehörigen Conforming-Programmkodesegment ist kleiner oder gleich der geforderten momentanen Privilegierungstufe (CPL) des unterbrochenen Programms.

10. Ein-/Ausgabe

Der Mikroprozessor 80286 ermöglicht den Zugriff auf Ein-/Ausgabe-Peripherie-Baugruppen, ähnlich wie bei fast allen anderen Mikroprozessorsystemen, über zwei unterschiedliche Verfahren:

- Nutzung des speziellen 80286-Ein-/Ausgabe-Adreßraums zur Ein-/Ausgabe,
- Nutzung des Speicheradreßraums zur Ein-/Ausgabe (memory mapped).

Welches der beiden Ein-/Ausgabe-Verfahren angewendet wird, hängt von den Möglichkeiten und Erfordernissen in einer konkreten 80286-Rechnerkonfiguration ab.
Außer diesen beiden Ein-/Ausgabe-Verfahren existiert natürlich auch im Mikroprozessorsystem 80286 die Möglichkeit, über externe Coprozessoren im DMA-Betrieb (direct memory access) auf den Ein-/Ausgabe-Adreßraum zuzugreifen.

10.1. Ein-/Ausgabe-Adreßraum

Bei der Nutzung des speziellen 80286-Ein-/Ausgabe-Adreßraums zum Datentransfer zwischen Mikrorechner und Peripheriebaugruppen erfolgt die Ein-/Ausgabe mit Hilfe der Ein-/Ausgabe-Befehle des 80286-Befehlssatzes.
Der Ein-/Ausgabe-Adreßraum besteht aus 65536 (64 K) einzeln adressierbaren 8-Bit-Ports.
Jeweils zwei hintereinanderliegende 8-Bit-Ports können bei diesem Verfahren auch verkettet zu einem 16-Bit-Port zusammengefügt werden, so daß insgesamt 32768 (32 K) 16-Bit-Ports oder 65536 (64 K) 8-Bit-Ports erreicht werden.
Die Portadressen 00F8H bis 00FFH sind reserviert, sie dürfen vom Programmierer nicht genutzt werden.
Teil der reservierten Portadressen sind der Adreßbereich 00F8H bis 00FCH. Sie dienen der Kommunikation zwischen 80286-CPU und dem Numerikcoprozessor 80287.
Da die Datentransferbreite zwischen dem Mikroprozessor 80286 und den Ein-/Ausgabe-Ports, wie erläutert, 8 oder 16 Bit betragen kann, muß, ähnlich wie beim Speicherzugriff, zwischen geraden und ungeraden Ein-/Ausgabe-Portadressen unterschieden werden.

Adresse	Zugriff	Beispiel
Gerade Wortadresse	16-Bit-Port	OUT 09EH,AX
Gerade Byteadresse	8-Bit-Port untere Hälfte (D7...D0) des 16-Bit-Datenstroms	IN AL,09EH
Ungerade Byteadresse	8-Bit-Port obere Hälfte (D15...D8) des 16-Bit-Datenstroms	OUT 09FH,AL

Der Zugriff auf den Ein-/Ausgabe-Adreßraum kann im RA-Mode ohne Restriktion erfolgen.
Im PVA-Mode sind die im Abschnitt 10.3. (Ein-/Ausgabe-Zugriffsschutz) folgenden Erläuterungen zu beachten.

10.2. Speicheradreßraum

Der Zugriff auf Ein-/Ausgabe-Peripherie-Baugruppen, deren Zugriffsadressen im Speicheradreßraum des Mikroprozessors 80286 angeordnet werden, erfolgt mit den für den Speicherzugriff bestimmten Befehlen und dem zugehörigen Hardwaresignalspiel.
Im PVA-Mode sind dann alle in dieser Betriebsart vorgesehenen Besonderheiten hinsichtlich des Zugriffsschutzes zu beachten (Abschn. 7. Zugriffsschutz).
Der gesamte Datenverkehr zwischen Ein-/Ausgabe-Peripherie-Baugruppen erfolgt bei diesen Memory-Mapped-Verfahren genau wie der Datentransfer mit Speicherbaugruppen.
Voraussetzung zur Anwendung des Memory-Mapped-Verfahrens ist die hardwaremäßige Realisierung des Speicherzugriffssignalspiels beim Anschluß von Ein-/Ausgabe-Baugruppen.

10.3. Ein-/Ausgabe-Zugriffsschutz

Erfolgt der Zugriff auf Peripheriebaugruppen im PVA-Mode über den Ein-/Ausgabe-Adreßraum, existiert beim Mikroprozessor 80286 ein spezieller Ein-/Ausgabe-Zugriffsschutzmechanismus. Er ermöglicht Ein-/Ausgabe-Operationen auf bestimmte Privilegierungsebenen zu begrenzen.
Ausgangspunkt des Ein-/Ausgabe-Zugriffsschutzes sind die beiden IOPL-Bits (input output privilege level) im Flagregister der 80286-CPU (Abschn. 4.4. Flagregister).
Diese Bits sind bei der Ausführung der Befehle

IN	Eingabe,
OUT	Ausgabe,
INS	Stringeingabe,
OUTS	Stringausgabe,
STI	Setzen Interruptflag (Interruptfreigabe) und
CLI	Löschen Interruptflag (Interruptsperre)

wirksam. Sie beschränken die Ein-/Ausgabe-Rechte einer Task.
Bei der Ein-/Ausgabe-Befehlsausführung muß dann die wichtige Bedingung erfüllt sein, daß die augenblickliche Privilegierungsstufe der Task (CPL) numerisch kleiner oder gleich der Ein-/Ausgabe-Privilegierungsstufe (IOPL) ist. Anderenfalls wird ein interner Fehlerinterrupt (general protection exception; Interruptvektor 13) erzeugt.

11. Datenformate

Die Basisdatentypen des Mikroprozessors 80286 sind das Byte (8 Bit) und das Wort (16 Bit). Ein Wort besteht dabei immer aus zwei aufeinanderfolgenden Bytes. Die Bitpositionen eines Bytes bzw. Wortes werden von Bit 0 (LSB least significant bit) bis Bit 7 bzw. 15 (MSB most significant bit) numeriert.
Das Byte eines Wortes, das die niederwertigen 8 Bit enthält, wird als Low-Byte bezeichnet, das Byte, das die Bits 8 bis 15 enthält, heißt High-Byte. Wird ein Wort im Speicher eines 80286-Mikroprozessors auf einer angegebenen Adresse abgelegt, befindet sich auf dieser Speicheradresse immer das Low-Byte und auf der folgenden Speicheradresse das High-Byte.
Worte können beim Mikroprozessor 80286 auf geraden oder ungeraden Adressen abgelegt werden. Dadurch lassen sich Datenstrukturen, die aus unterschiedlichen Elementen bestehen, dicht packen. Zu beachten ist jedoch, daß bei einem Speicherzugriff auf ein Wort, das auf einer ungeraden Speicheradresse abgelegt ist, zwei Speicherzyklen erforderlich sind (ein Bytezugriff für das Low-Byte und ein Bytezugriff für das High-Byte). Bei einem Speicherzugriff auf ein Wort, dessen Low-Byte sich auf einer geraden Speicheradresse befindet, wird im Gegensatz dazu nur ein Speicherzyklus benötigt. Deshalb sollten Worte, auf die im Programm häufig zugegriffen wird, in Datenstrukturen möglichst auf geraden Speicheradressen vereinbart werden, um einen hohen Systemdurchsatz zu erreichen.
Alle weiteren Datentypen, die vom Mikroprozessor 80286 unterstützt werden, lassen sich auf die Basisdatentypen Byte und Wort zurückführen. Bild 11.1 gibt eine Übersicht über die Datenformate.

Der Mikroprozessor 80286 unterscheidet:

- vorzeichenlose Zahlen oder Binärwerte (unsigned integer),
- vorzeichenbehaftete Zahlen (integer),
- BCD-Zahlen (BCD binary coded decimal),
- Zeichenketten (strings),
- Pointer,
- Gleitkommazahlen (floating point numbers).

Vorzeichenlose Zahlen (unsigned integer) können im Byte-, Wort- oder Doppelwortformat genutzt werden. Das Doppelwortformat tritt nur als Ergebnis der Multiplikation zweier Wortgrößen in einem Doppelregister auf bzw. kann als Dividend benutzt werden. Dieses Datenformat erhält seine Daseinsberechtigung erst in Verbindung mit einem Numerikcoprozessor.
Alle Bitpositionen in dem Byte, Wort oder Doppelwort, das eine vorzeichenlose Zahl repräsentiert, bestimmen den Betrag dieser Zahl. Im Datenformat unsigned integer sind die folgenden Zahlenbereiche verfügbar:

Byte:	0	255	(2^8 - 1),
Wort:	0	65535	(2^16 - 1),
Doppelwort:	0 ...	4294967295	(2^32 - 1).

Das Datenformat unsigned integer wird auch zur Speicherung von binären Informationen (z.B. Bitvariablen, Statusworte) benutzt.

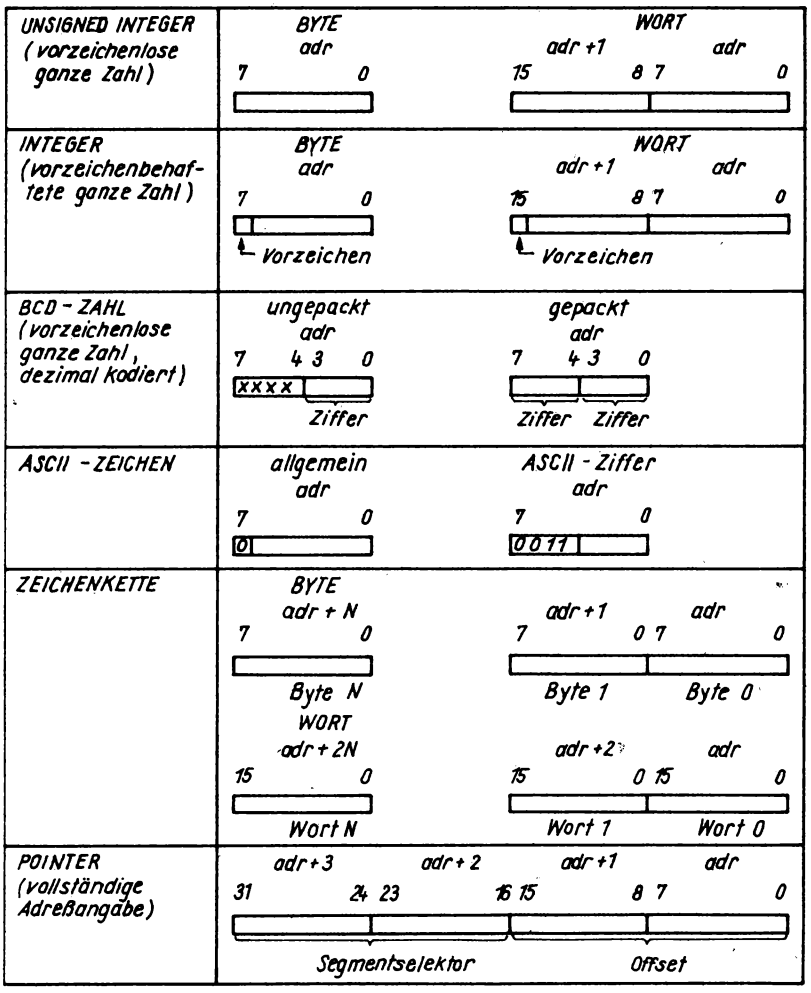


Bild 11.1. Datenformate des Mikroprozessors 80286

Bei vorzeichenbehafteten Zahlen (integer), die im Byte-, Wort- oder Doppelwortformat verwendet werden können, repräsentiert das höchstwertige Bit (MSB) im Byte, Wort oder Doppelwort das Vorzeichen des Zahlenwerts. Auch hier gilt, daß Integerzahlen im Doppelwortformat in Systemen ohne Numerikcoprozessor nur als Ergebnis einer Multiplikation zweier Wortgrößen in einem Doppelregister entstehen können bzw. als Dividend benutzt werden.
Negative Integerwerte werden beim 80286 wie bei allen anderen Mikroprozessoren im Zweierkomplement dargestellt.

```
MSB = 0      positiv
MSB = 1      negativ
```

Vorzeichenbehaftete Zahlen überstreichen dann den folgenden Zahlenbereich:

```
Byte:                -128           +127   (2^7 - 1),
Wort:              -32768         +32767   (2^15 - 1),
Doppelwort:   -2147473648    +2147483647   (2^31 - 1).
```

Die Zahl Null ist immer positiv.
Soll eine vorzeichenbehaftete Integerzahl im Byte- bzw. Wortformat in ein vorzeichenbehaftetes Integerformat im Wort- bzw. Doppelwortformat konvertiert werden, ist das Vorzeichenbit in jede Bitposition des höherwertigen Datenbytes bzw. Datenworts zu übertragen, um im neuen Datenformat den gleichen Wert zu repräsentieren. Das folgende Beispiel verdeutlicht diesen Algorithmus:

```
Konvertierung:      Byte -> Wort

  54H           01010100 ->                  00000000 01010100      0054H
  88H           10001000 ->                  11111111 10001000      FF88H

Konvertierung:      Wort -> Doppelwort

1234H  00010010 00110100 -> 00000000 00000000 00010010 00110100  00001234H
A123H  10100001 00100011 -> 11111111 11111111 10100001 00100011  FFFFA123H
```

Der Mikroprozessor 80286 unterstützt diese beiden Konvertierungen durch spezielle Befehle (CBW Konvertierung Byte in Wort, CWD Konvertierung Wort in Doppelwort).

BCD-Zahlen (binary coded decimal) werden für spezielle dezimale Anzeigeelemente (z.B. 7-Segment-Anzeigebauelemente), besonders aber auch in hochgenauen Arithmetikpaketen benutzt. Bei BCD-Zahlen werden die Ziffern 0 bis 9 in vier Bitpositionen eines Bytes kodiert. BCD-Zahlen sind immer vorzeichenlos. Sie können in gepacktem Format (zwei BCD-Ziffern in einem Byte) oder im ungepackten Format (eine BCD-Ziffer auf den Bitpositionen 0 bis 3 eines Bytes, Bit 4 bis 7 unbenutzt oder null) genutzt werden.
Bei arithmetischen Operationen mit BCD-Zahlen sind zwei Phasen notwendig - die Berechnung und die Korrektur. Für die Korrektur stehen dabei sechs spezielle Korrekturbefehle zur Verfügung.
Ungepackte BCD-Zahlen können beim 80286 addiert, subtrahiert, multipli-

ziert und dividiert werden. Für gepackte BCD-Zahlen stehen nur die Addition und die Subtraktion zur Verfügung. Die Befehle zur Addition, Subtraktion und Multiplikation ungepackter BCD-Zahlen liefern ein Zwischenergebnis, das anschließend korrigiert werden muß. Bei der Division wird die ungepackte BCD-Zahl zuerst korrigiert und anschließend der Divisionsbefehl ausgeführt. Nachfolgend werden die Zuordnungen der Korrekturbefehle zu den entsprechenden Arithmetikbefehlen wiedergegeben.

BCD-Format	Rechenart	Befehlsreihenfolge
Ungepackt	Addition	ADD AL,src AAA
	Subtraktion	SUB AL,src AAS
	Multiplikation	MUL src AAM
	Division	AAD DIV src
Gepackt	Addition	ADD AL,src DAA
	Subtraktion	SUB AL,src DAS

ASCII-Zeichen, die die Ziffern 0 bis 9 repräsentieren, können arithmetisch wie ungepackte BCD-Ziffern verarbeitet werden. Es ist jedoch darauf zu achten, daß vor Multiplikations- und Divisionsbefehlen die Bitpositionen 4 bis 7 auf Null gesetzt werden (z.B. durch AND AL,0FH). Nach erfolgter Berechnung kann das Ergebnis problemlos wieder in ein ASCII-Zeichen umgewandelt werden (z.B. durch OR AL,30H).

Neben Befehlen zur Behandlung von Zahlenwerten verfügt der Mikroprozessor 80286 natürlich auch über leistungsfähige Befehle zur Manipulation von Zeichenketten (strings). Die Elemente einer Zeichenkette können Bytes oder Worte sein. Die Länge einer Zeichenkette darf in beiden Betriebsarten (RA-Mode und PVA-Mode) ein Speichersegment (64 KByte) nicht überschreiten.

Pointer sind ein eigenständiges Datenformat. Sie sind als Zeiger auf Speicheradressen zu verstehen und bestehen aus einem Segmentselektor und einem Offset (jeweils 16 Bit umfassend). Beide Wortgrößen werden im Speicher eines 80286-Mikrorechners immer in der Reihenfolge zuerst Offset (Adresse n) und anschließend Segmentselektor (Adresse n+2) abgelegt.

Gleitkommazahlen sind nur in Verbindung mit dem Arithmetikcoprozessor 80287 sinnvoll, der verschiedene erweiterte Rechenarten mit Integer-, Gleitkomma- und BCD-Formaten unterstützt. Das Gleitkommaformat des 80286/80287 entspricht dem IEEE-Standard 754 für Gleitkommazahlen.

12. Adressierungsarten

Viele Befehle des Mikroprozessors 80286 arbeiten mit Operanden, die direkt im Befehl angegeben werden können (Direktoperanden) oder sich in Registern (Byte-, Wort- oder Steuerregister), im Speicher oder auf Ein-/Ausgabe-Kanälen befinden.
Direktoperanden und Registeroperanden werden besonders schnell verarbeitet, da sie sich nach dem Einlesen des Befehls bereits im Prozessor befinden.
Für Speicheroperanden muß nach der Befehlsdekodierung erst ein Speicherlesezyklus eingeleitet werden, bevor deren Wert für die Verarbeitung zur Verfügung steht.
Ein Speicheroperand wird durch einen Pointer, bestehend aus dem Segmentselektor und dem Offset, adressiert. Es existieren jedoch nur wenige Befehle, die zur Operandenadressierung solch einen vollständigen 32-Bit-Wert im Befehlskode benötigen. In der Regel wird der Segmentselektor aus einem der vier Segmentregister genutzt. Im Befehl müssen dann nur das jeweilige Segmentregister, das Offset und ein eventueller Distanzwert angegeben werden.
Bei vielen Befehlen des Mikroprozessors 80286 besteht eine implizite Zuordnung von Segmentregister und Operandentyp.
Im allgemeinen beziehen sich alle Programmkodereferenzen auf das Segmentregister CS, alle Datenreferenzen (außer Referenzen über das Basisregister BP) auf das Datensegmentregister DS und alle Stackreferenzen (Befehle PUSH, POP) sowie Referenzen über das Basisregister BP auf das Stacksegmentregister SS.
Diese implizite Zuordnung der Segmentregister zu den Operandentypen kann jedoch mit Hilfe des Segment-Override-Präfixes bei Bedarf aufgehoben werden. Im Befehl wird dann vor den Operanden das gewünschte Segmentregister angegeben. Für das Segment-Override-Präfix wird in der Assemblersprache des Mikroprozessors 80286 folgende Schreibweise angewendet:

```
Segmentregister:Adresse          Beispiel:    mov   AX,ES:label
```

Im obigen Beispiel wird die Marke label über das Segmentregister ES adressiert.
Die Anwendung des Segment-Override-Präfixes ist architekturbedingt in den folgenden beiden Fällen nicht möglich:

- Bei Zeichenkettenoperationen muß die Zieladresse immer über das Segmentregister ES angegeben werden.

- Operanden, die mit Hilfe des Stackpointerregisters SP adressiert werden, sind nur über das Stacksegmentregister SS erreichbar.

Die Offsetkomponente gibt beim Mikroprozessor 80286 grundsätzlich den Versatz einer Speicheradresse innerhalb eines Segments bezogen auf den Segmentanfang an. Sie kann je nach Adressierungsart als Summe aus den folgenden drei Teilkomponenten gebildet werden:

- Im Befehl kann ein Distanzwert (displacement) angegeben werden, der entweder Byte- oder Wortgröße besitzt. Ein 8-Bit-Distanzwert wird intern vorzeichenbehaftet in einen 16-Bit-Wert umgewandelt. Distanzwerte mit einer Breite von 8 Bit sind bei der direkten

Adressierung (siehe unten) nicht zugelassen.

- Die Offsetkomponente ergibt sich aus dem Inhalt eines der beiden Basisregister (BX oder BP).

- Die Offsetkomponente ergibt sich aus dem Inhalt eines der beiden Indexregister (SI oder DI).

Jede dieser Komponenten kann einen positiven oder einen negativen Wert beinhalten. Die Berechnung des Offsets erfolgt immer modulo 65536, so daß ein Segment nie verlassen werden kann. Das aus den Komponenten resultierende Offset wird als effektive Adresse (EA) der Speichervariablen bezeichnet.
Bei Speicheroperanden unterstützt der Mikroprozessor 80286 sechs Adressierungsarten. Sie unterscheiden sich nur in der Berechnungsmethode der effektiven Adresse einer Speichervariablen (der resultierenden Offsetkomponente) innerhalb des ausgewählten Segments. Die Adressierungsart ist i.allg. im ModRM-Byte des Befehls (Abschn. 13.1. Befehlsaufbau) verschlüsselt. Es folgt eine Zusammenstellung der Adressierungsarten und der zugehörigen Berechnungsmethoden für den Mikroprozessor 80286:

Adressierungsart	Offsetberechnungsmethode
Direkte Adressierung	EA = disp
Indirekte Adressierung	EA = (BX) or (BP) or (SI) or (DI)
Basisadressierung	EA = [(BX) or (BP)] + disp
Indizierte Adressierung	EA = [(SI) or (DI)] + disp
Indizierte Basisadressierung	EA = [(BX) or (BP)] + [(SI) or (DI)]
Indizierte Basisadressierung mit Distanzwert	EA = [(BX) or (BP)] + [(SI) or (DI)] + disp

Die Adressierungsarten für Speichervariable sollen in der Folge ausführlich erläutert werden. Zur Adressierung einfacher Variablen (Zahlenwerte, Einzelzeichen) sind die ersten beiden Adressierungsarten günstig anwendbar:

Direkte Adressierung

Die Offsetkomponente der Operandenadresse wird direkt im Befehl als vorzeichenbehafteter 16-Bit-Wert angegeben.

Beispiel:

```
MOV   CX, VAR1  ;CX := Inhalt VAR1
```

Indirekte Adressierung

Die Offsetkomponente zur Adreßberechnung ist in einem der Indexregister (SI oder DI) oder in dem Basisregister BX enthalten. Das Feld für den Distanzwert (Bild 13.1) wird nicht benutzt.

Das Basisregister BP ist für die indirekte Adressierung nicht geeignet, da es als Basisregister im Stacksegment zur Adressierung lokaler, temporärer Variablen genutzt wird und somit zusätzlich immer eine Distanzangabe erfordert.

Beispiel:

```
MOV   CX, [BX]     ;BX enthält effektive Adresse
                   ;der zu ladenden Speichergröße
```

Die folgenden Adressierungsarten ermöglichen den Zugriff auf einzelne Elemente komplexer Datenstrukturen, die sich im Speicher befinden:

Basisadressierung

Die Offsetkomponente ergibt sich aus der Summe des Inhalts eines der beiden Basisregister (BX oder BP) und einem Distanzwert (8 oder 16 Bit), der im Befehl angegeben ist.
Die Basisadressierung wird benutzt, wenn in verschiedenen Datenstrukturen gleichen Typs, deren Basisadresse bekannt ist, auf das gleiche Datenelement zugegriffen werden soll. Das Basisregister adressiert dabei den Anfang der Datenstruktur, der Distanzwert gibt den Versatz des Datenelements bezogen auf den Strukturanfang an. Durch Veränderung der Basis, d.h. des Inhalts des Basisregisters, kann so auf korrespondierende Datenelemente schnell und unkompliziert zugegriffen werden.

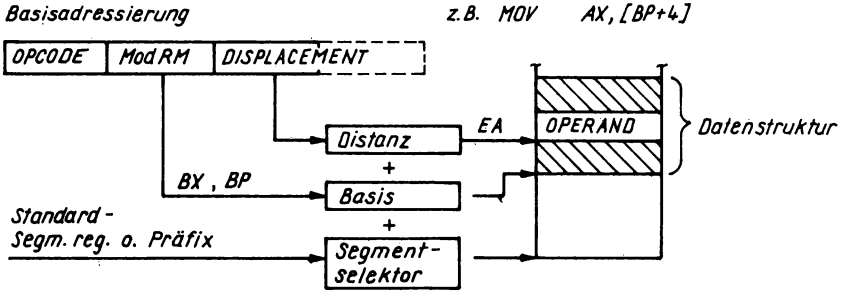


Bild 12.1. Prinzip der Basisadressierung

Indizierte Adressierung

Die Offsetkomponente ergibt sich aus der Summe des Inhalts eines Indexregisters (SI oder DI) und aus einem im Befehl angegebenen Distanzwert (8 oder 16 Bit).
Die indizierte Adressierung ist besonders vorteilhaft in Datenfeldern oder Datenstrukturen anwendbar, die aus Elementen gleicher Größe bestehen. In diesem Fall adressiert der Distanzwert die Startadresse

des Feldes, und das Indexregister enthält den Versatz des Datenelements. Durch einfache arithmetische Operationen mit dem Inhalt des Indexregisters können nacheinander die einzelnen Elemente des Feldes oder der Struktur adressiert werden. Die komfortablen Zeichenkettenmanipulationsbefehle des Mikroprozessors 80286 nutzen diese Adressierungsart in besonders markanter Weise.

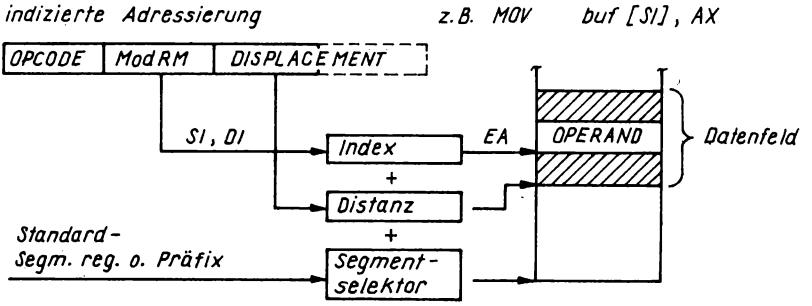


Bild 12.2. Prinzip der indizierten Adressierung

Indizierte Basisadressierung

Wie aus der Bezeichnung schon hervorgeht, stellt diese Adressierungsart eine Verknüpfung der Basisadressierung und der indizierten Adressierung dar.
Die Offsetkomponente wird als Summe der Inhalte von Basis- und Indexregister und eines wahlweise direkt angegebenen Distanzwerts berechnet. Die indizierte Basisadressierung ermöglicht einen effektiven Zugriff auf einzelne Elemente eines Feldes innerhalb einer Struktur.

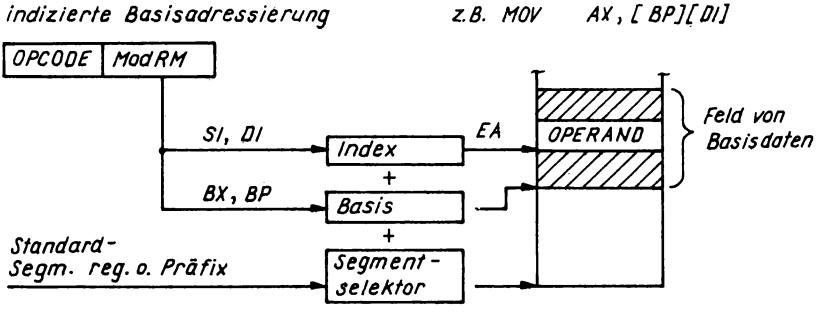


Bild 12.3. Prinzip der indizierten Basisadressierung (ohne Distanzwertangabe)

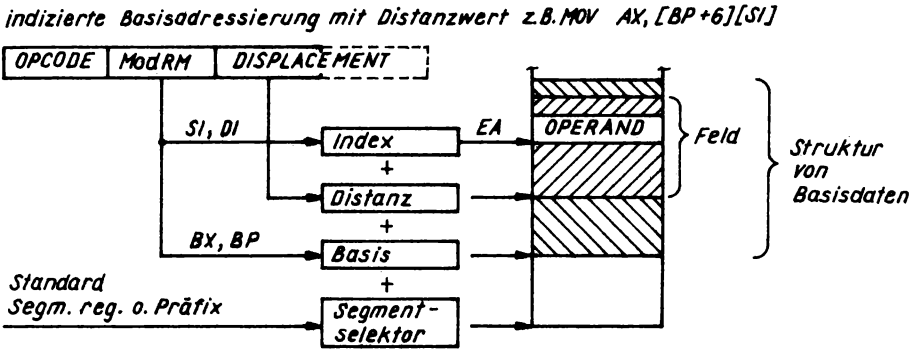


Bild 12.4. Prinzip der indizierten Basisadressierung (mit Distanzwertangabe)

13. Befehlssatz

In diesem Abschnitt wird der Befehlssatz des Mikroprozessors 80286 detailliert beschrieben.
Der Aufbau der 80286-Befehle ist nicht so regelmäßig wie bei anderen Mikroprozessoren und bedarf deshalb einer ausführlichen Erläuterung (Abschn. 13.1. Befehlsaufbau). Die einzelnen Befehlsbeschreibungen sind zur besseren Handhabung alphabetisch geordnet. Eine logische Gruppierung der verschiedenen Befehlstypen ist aus der Übersicht im Abschnitt 13.2.2. Befehlssatzübersicht ersichtlich.
In der detaillierten Befehlsbeschreibung (Abschn. 13.3.) werden für jeden Befehl der hexadezimale Operationskode, der dem Befehl zugeordnete mnemonische Befehlskode mit allen möglichen Operandenkombinationen, die Anzahl der Takte, die für die Abarbeitung des Befehl benötigt werden, sowie eine Kurzcharakteristik der auszuführenden Aktionen angegeben. Erläuterungen zum Aufbau der Befehlsbeschreibungen, zu Begriffen und Abkürzungen werden im Abschnitt 13.2.1. gegeben.

Im Anhang A ist eine vollständige Befehlsliste enthalten.

13.1. Befehlsaufbau

Befehle des Mikroprozessors 80286 bestehen aus einem bis maximal sechs Byte. Im Bild 13.1. ist der allgemeine Befehlsaufbau dargestellt.

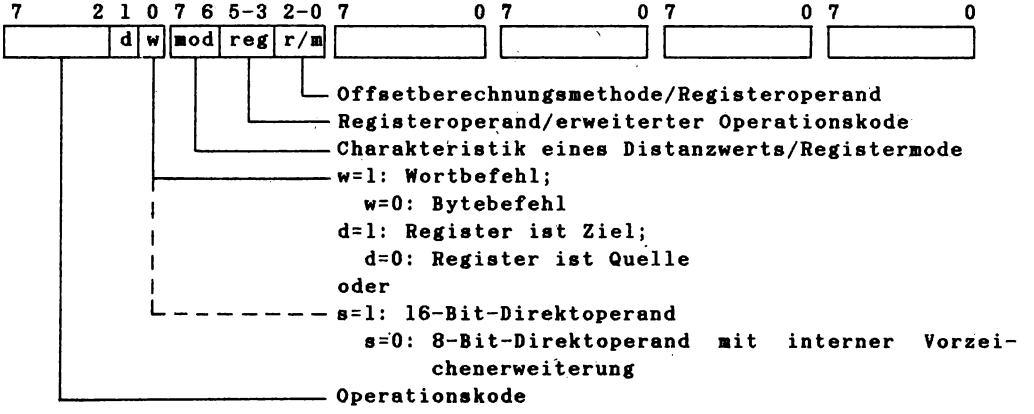


Bild 13.1. Allgemeiner Befehlsaufbau des Mikroprozessors 80286

Das erste Byte enthält immer den Operationskode (auch als Opkode bezeichnet). Bei einigen Befehlen besteht der Operationskode auch aus zwei Bytes. Im Operationskode werden verschiedene Informationen verschlüsselt:

- der Befehlstyp,
- Informationen über die folgenden Bytes des Befehls (Direktwerte, implizite Registeradressen),
- die Unterscheidung zwischen Byte- und Wortbefehlen (w word),
- eine Richtungsinformation für einen Registeroperanden (d direction),
- eine Information über die Benutzung von Direktwerten (s sign extension),
- eine Registeradresse (nur bei einigen Befehlen).

Bei vielen Befehlen folgt nach dem Operationskode ein sogenanntes "ModRM-Byte" (modus register memory). Es dient der weiteren Spezifikation der Operation bzw. der Operanden. Nicht alle Befehle des 80286 benötigen ein ModRM-Byte (z.B. bedingte Sprungbefehle, einige Operationen mit dem Akkumulator).

Distanzwerte zur Adreßbildung werden in der Reihenfolge Low-Byte vor High-Byte in den letzten Bytes des Befehls angegeben. Bei 8-Bit-Distanzwerten entfällt das High-Byte.

Direktwerte folgen sofort nach dem ModRM-Byte oder, falls dieses nicht vorhanden ist, sofort nach dem Operationskode. Sie werden also immer vor einem eventuellen Distanzwert angegeben.
Das ModRM-Byte enthält drei Bitfelder, in denen folgende Informationen verschlüsselt sind:

mod-Feld

In den Bitpositionen 7 und 6 wird das Vorhandensein bzw. die Länge eines Distanzwerts angegeben.

mod=00 Im Befehl wird kein Distanzwert benutzt (z.B. bei indizierter Basisadressierung). Die Bytes disp-low und disp-high sind nicht vorhanden.

Ausnahme:
Sind im ModRM-Byte die Felder mod=00 und r/m=110 (diese Kodierung entspräche der unzulässigen indirekten Adressierung über Basisregister BP), ergibt sich die effektive Adresse aus einem 16-Bit-Distanzwert disp-high:disp-low.

mod=01 Der angegebene 8-Bit-Distanzwert (disp-low) wird intern vorzeichenbehaftet auf Wortgröße erweitert und zur Offsetberechnung benutzt. Das Element disp-high ist nicht vorhanden.

mod=10 Der 16-Bit-Distanzwert ergibt sich aus den Bytes
disp-high:disp-low.

mod=11 Bei dieser Kodierung wird ebenfalls kein Distanzwert verwendet. Sie kennzeichnet Befehle, mit denen zwei Registeroperanden verarbeitet werden sollen. Das "r/m"-Feld des ModRM-Bytes wird dann zur Adressierung des zweiten Registers benutzt.

reg-Feld

Die Bits 5, 4 und 3 des ModRM-Bytes können auf verschiedene Weise interpretiert werden:

- zur Angabe eines Registeroperanden (r-Format) oder
- zur weiteren Spezifikation des Befehlskodes (n-Format).

Die Unterscheidung zwischen beiden Interpretationen erkennt der Mikroprozessor anhand des Operationskodes. Sie ist für jeden Befehl im Mikroprogramm fest vorgegeben.
Im r-Format wird die Registeradresse des Registeroperanden im "reg"-Feld kodiert. Die nachfolgende Tafel gibt die Kodiervorschrift an:

Tafel 13.1. Kodierung der Register des Mikroprozessors 80286

Kodierung	Byteregister	Wortregister	Segmentregister
0	000 AL	000 AX	00 ES
1	001 CL	001 CX	01 CS
2	010 DL	010 DX	10 SS
3	011 BL	011 BX	11 DS
4	100 AH	100 SP	
5	101 CH	101 BP	
6	110 DH	110 SI	
7	111 BH	111 DI	

In der ausführlichen Befehlsbeschreibung (Abschn. 13.3.) werden Befehle, die ein ModRM-Byte im r-Format benutzen, nach dem hexadezimalen Operationskode mit den Zeichen "/r" gekennzeichnet.
Befehle, die das n-Format des ModRM-Bytes verwenden, verarbeiten meist nur einen Register- oder Speicheroperanden oder der zweite Operand ist bereits im Operationskode verschlüsselt. Somit steht das "reg"-Feld zur weiteren Spezifikation des Operationskodes zur Verfügung. Das n-Format wird in der ausführlichen Befehlsbeschreibung durch einen Schrägstrich und eine Ziffer (0 bis 7) nach dem Operationskode gekennzeichnet. Die Ziffer entspricht dabei der Kodierung, die im "reg"-Feld des ModRM-Bytes eingetragen wird.

```
Beispiel:      FE /1    DEC eb     entspricht
               1111111O mo 001 r/m
```

r/m-Feld

Im letzten Feld des ModRM-Bytes (Bit 2 bis 0) wird entweder die Berechnungsmethode zur Bildung der effektiven Adresse eines Speicheroperanden ("mod"-Feld ungleich 11) oder eine zweite Registeradresse verschlüsselt ("mod"-Feld gleich 11). Für die Verschlüsselung eines Registeroperanden gilt die Kodiervorschrift lt. Tafel 13.1.

Die folgende Tabelle gibt die Kodierungen für die unterschiedlichen Offsetberechnungsmethoden an.

r/m	Offsetberechnungsmethode
000	(BX) + (SI) + Distanz
001	(BX) + (DI) + Distanz
010	(BP) + (SI) + Distanz
011	(BP) + (DI) + Distanz
100	(SI) + Distanz
101	(DI) + Distanz
110	(BP) + Distanz (siehe Ausnahme mod=00)
111	(BX) + Distanz

Bei einigen Befehlen des Mikroprozessors 80286 können Registeroperanden bereits im ersten Byte des Operationskodes verschlüsselt werden, wobei das ModRM-Byte entfällt. Diese Befehle sind in der Befehlsbeschreibung durch die Zeichen "+rw" nach dem hexadezimalen Operationskode gekennzeichnet. Die Verschlüsselung der Registeroperanden erfolgt durch Addition des entsprechenden Registerkodes (s. Tafel 13.1.) zum Befehlskode.

Im Anhang B ist eine Tabelle enthalten, in der alle Kodierungen des ModRM-Bytes zusammengefaßt sind.

13.2. Befehle des Mikroprozessors 80286

13.2.1. Aufbau der Befehlsbeschreibung

Nachfolgend ist ein charakteristischer Eintrag für eine Befehlsbeschreibung dargestellt:

```
DEC                          Dekrementieren um 1                          DEC

Opkode     Befehl       Takte       Beschreibung

FE  /1     DEC eb       2,mem=7     eb := eb - 1
FF  /1     DEC ew       2,mem=7     ew := ew - 1.
48+ rw     DEC rw       2           rw := rw - 1

Flags:     OF DF IF TF SF ZF AF PF CF
           M  -  -  -  M  M  M  M  -

Operation:
    Vom Inhalt des Zieloperanden wird 1 subtrahiert. Das Flag CF wird
    nicht beeinflußt. Soll das Flag CF beeinflußt werden, ist der
    Befehl SUB mit dem Direktoperanden 1 zu benutzen.

Exceptions im PVA-Mode:
#GP(0)          Der Operand befindet sich in einem schreibgeschützten
                Segment.
#GP(0)          Die effektive Adresse eines Speicheroperanden in einem
                der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)          Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13    Der Wortoperand besitzt ein Offset von OFFFFH.
```

Unter der Spalte Opkode wird die hexadezimale Kodierung des Operationskodes angegeben. Die durch einen Schrägstrich gekennzeichneten Angaben beschreiben den Aufbau eines ModRM-Bytes (Abschn. 13.1. Befehlsaufbau). Nach dem hexadezimalen Operationskode folgt in der Spalte Befehl der mnemonische Operationskode mit der entsprechenden Operandenkombination. Für die verschiedenen Operandentypen werden folgende Abkürzungen benutzt:

Programmkodeadressen

Programmkodeadressen werden als Operanden bei Sprungbefehlen, Schleifensteuerbefehlen und Unterprogammaufrufen benutzt. Sie können als Distanzwert zur Berechnung des Offsets innerhalb des aktuellen Programmkodesegments oder als vollständige Speicheradresse angegeben werden.

cb	Distanzwert 8 Bit vorzeichenbehaftet
cw	Distanzwert 16 Bit vorzeichenbehaftet
cd	vollständige Speicheradresse (Segmentselektor:Offset)

Unter Benutzung von Distanzwerten ergibt sich die effektive Speicheradresse (z.B. der neue Wert des Instruktionspointers bei einem relativen Sprungbefehl) aus der Summe des Offsetwerts des Befehls, der dem Sprungbefehl folgt, und dem Distanzwert selbst. Die Addition erfolgt immer modulo 65536, d.h., das aktuelle Programmkodesegment kann nicht verlassen werden. Es ist zu beachten, daß auch die 16-Bit-Distanzwerte vorzeichenbehaftet sind. Distanzen größer als 32767 werden deshalb durch negative Werte repräsentiert.
Distanzwerte mit einer Breite von nur 8 Bit (Form cb) werden vor der Addition vorzeichenrichtig zu einen 16-Bit-Wert erweitert, indem das Bit 7 des Distanzbytes in jede Bitposition des höherwertigen Bytes übertragen wird. Bei allen bedingten Sprungbefehlen und bei den Schleifensteuerbefehlen des Mikroprozessors 80286 werden nur 8-Bit-Distanzwerte benutzt. Mit diesen Befehlen kann also nur ein Sprungziel im Bereich von -128 bis +127 Byte, ausgehend von der aktuellen Programmkodeadresse, erreicht werden. Für bedingte Sprünge zu Adressen außerhalb dieser Distanz ist die sogenannte "jump-around-jump"-Methode anzuwenden (s. Befehlsbeschreibung der bedingten Sprungbefehle).
Vollständige Programmkodeadressen werden durch einen 4-Byte-Pointer angegeben, der einen kompletten Adreßwert CS:IP repräsentiert. Im hexadezimalen Befehlskode wird dabei zuerst die Offsetkomponente angegeben, dann der Segmentselektor.

Direktoperanden

Direktoperanden werden in Befehlen zur Wertzuweisung benutzt.

db	Direktwert 8 Bit breit
dw	Direktwert 16 Bit breit

Direktoperanden in Bytegröße können in Befehlen mit Wortoperanden kombiniert werden. Dazu wird der Byteoperand in der CPU vorzeichenrichtig auf Wortgröße erweitert (s. Abschn. 11. Datenformate).

Registeroperanden

Registeroperanden dienen zur Adressierung der internen allgemeinen Register des Mikroprozessors 80286.

rb	Byteregister
rw	Wortregister

Registeroperanden werden meist im ModRM-Byte verschlüsselt. Die Unterscheidung zwischen Byte- und Wortregistern wird durch das Bit 0 des ersten Operationskodebytes bestimmt. Registeroperanden können bei einigen Befehlen jedoch auch direkt im Operationskodebyte verschlüsselt sein (Abschn. 13.1. Befehlsaufbau).

Effektive Operandenadressen

Über effektive Operandenadressen können sowohl Speicheroperanden als auch Registeroperanden spezifiziert werden. Der Typ des Operanden

wird durch das ModRM-Byte bestimmt. Für Speichervariable sind alle Adressierungsarten zulässig, d.h., die Speichervariable kann auch über Basis- oder Indexadressierung erreicht werden.

eb	effektive Adresse eines Byteoperanden
ew	effektive Adresse eines Wortoperanden
ed	speicherbezogener Pointeroperand

Der Operandentyp ed stellt eine vollständige Speicheradresse, bestehend aus Segmentselektor und Offset, dar. Die Adresse (d.h. der Pointer) muß jedoch im Speicher abgelegt sein. Wird im Befehl fälschlicherweise eine Registervariable adressiert, löst der Prozessor die Exception 6 aus (Abschn. 9.3. Reservierte Interruptvektoren).

Speicheroperanden

Speicheroperanden können im Gegensatz zu effektiven Operandenadressen nur Speichervariablen adressieren. Dabei sind alle Adressierungsarten zulässig (auch Basis- oder Indexadressierung). Die unzulässige Adressierung von Registervariablen führt zur Exception 6.

mb	Speicheroperand in Bytegröße
mw	Speicheroperand in Wortgröße
m	Speicheradresse allgemein

Die folgenden Operandentypen adressieren ebenfalls Speichervariablen, jedoch sind die Adressierungsarten Basisadressierung oder indizierte Adressierung wie auch deren Verknüpfung nicht zulässig. Diese optimierte Form spart Speicherplatz im Befehlskodebereich, ist jedoch nur bei einigen Befehlen des Mikroprozessors 80286 anwendbar.

xb	Speicheroperand in Bytegröße
xw	Speicheroperand in Wortgröße

In der Befehlsliste ist für jeden Befehl die Zeitdauer der Befehlsausführung (Anzahl der Takte) angegeben. Als Taktfrequenz wird hier die interne Arbeitsfrequenz des Mikroprozessors bezeichnet. Sie entspricht genau der Hälfte der Frequenz, die am Pin CLK des Mikroprozessors angelegt wird.
Die angegebene Taktanzahl, die zur Befehlsausführung benötigt wird, erhöht sich unter folgenden Bedingungen:

1 Takt bei Speicherzugriffen, bei denen zur Adreßbildung drei Komponenten, d.h., sowohl eine Basisadresse als auch ein Index und ein Distanzwert benutzt werden (indizierte Basisadressierung mit Distanzwert),

2 Takte bei wortorientierten Speicherzugriffen auf Operanden, die auf ungeraden Adressen liegen (die Bytes des Wortoperanden werden in diesem Fall einzeln gelesen),

1 Takt für jeden WAIT-Zyklus. WAIT-Zyklen beeinflussen nur die Zeitdauer des Befehlslesezyklus. WAIT-Zyklen müssen nicht unbedingt eine Verlängerung der Befehlsausführung bewirken, da der Mikro-

prozessor 80286 mit einer Befehlswarteschlange in einem internen Cacheregister arbeitet (prefetching),

n Takte nach Ausführung eines Programmsteuerbefehls. Da bei Befehlen dieser Kategorie die interne Befehlswarteschlange des 80286 geleert wird, ist für jedes Byte des folgenden Befehls ein Takt zu addieren.

Die angegebene Taktanzahl stellt die minimale Befehlsausführungszeit dar. Da einige Befehlsfolgen schneller ausgeführt werden, als sie vom Mikroprozessor 80286 gelesen werden können, sind selbst bei Buszyklen ohne Verzögerungen 5 bis 10% der ermittelten Rechenzeit zu addieren.

Bei einigen Befehlen sind zwei Taktangaben vorhanden, die für verschiedene Bedingungen gelten:

mem= Der angegebene Operand kann sowohl eine Registervariable als auch eine Speichervariable repräsentieren. Die erstgenannte Taktangabe steht für die Registervariable, die zweite (mem=) für die Speichervariable. Bei Anwendung der indizierten Basisadressierung mit Distanzwert (Adreßbildung mit drei Komponenten) ist ein Takt zu addieren.

noj= Diese Angabe steht bei bedingten Sprungbefehlen oder bei Interruptbefehlen. Die erste Taktangabe gilt, wenn der Sprung ausgeführt wird, die zweite, wenn der dem Sprungbefehl folgende Befehl abgearbeitet wird (kein Sprung).

pm= Bei diesen Befehlen ist die Ausführungszeit von der Betriebsart des Mikroprozessors abhängig. Die erste Angabe gilt im RA-Mode, die zweite im PVA-Mode.

In der Befehlsliste werden zu jedem Befehl die möglichen Veränderungen des Flagregisters angegeben. Dabei werden die folgenden Abkürzungen verwendet:

M	Wert des Flagbits wird durch den Befehl modifiziert.
U	Wert des Flagbits ist undefiniert.
0	Flagbit wird auf 0 gesetzt.
1	Flagbit wird auf 1 gesetzt.
0	Flagbit bleibt unverändert.

Am Schluß jeder Befehlsbeschreibung werden mögliche Exceptions beschrieben, die bei der Abarbeitung des Befehls auftreten können. Exceptions im PVA-Mode werden durch ein Doppelkreuz und zwei Großbuchstaben gekennzeichnet, denen in runden Klammern ein Fehlerkode folgen kann. Die Wirkungen der einzelnen Exceptions sowie die verwendeten Abkürzungen wurden bereits im Abschnitt 9.3. Reservierte Interruptvektoren ausführlich erläutert.

Für die Entwicklung von Betriebssystemprogrammen, in denen die leistungsfähige Systemarchitektur des 80286 im PVA-Mode möglichst effektiv ausgenutzt werden soll, ist die genaue Kenntnis der internen Abläufe im Prozessorschaltkreis wichtig. Deshalb wird in der Befehlsbeschreibung bei jenen Befehlen, die direkten Bezug auf das Speicherverwaltungssystem, das

Interruptsystem oder die implementierten Taskwechselmechanismen haben, ein Algorithmus zur Verdeutlichung der Reihenfolge der internen Aktionen angegeben. Die hier verwendete Notation ist frei gewählt, lehnt sich aber an übliche höhere Programmiersprachen an.

Im RA-Mode, in dem die hochentwickelten Schutzmechanismen des Mikroprozessors 80286 nicht wirksam sind, existieren nur wenige Exceptions.

13.2.2. Befehlssatzübersicht

Die Übersicht enthält alle Befehlstypen des Mikroprozessors 80286. Sie sind in logisch zusammenhängenden Gruppen gegliedert. Die Seitennummer verweist auf die detaillierte Befehlsbeschreibung.

Befehl	Kurzcharakteristik	Seite
Datentransferbefehle		
MOV	Lade Byte/Wort	170
PUSH	Wort im Stack ablegen	180
POP	Wort aus Stack holen	177
PUSHA	PUSH alle Register	180
POPA	POP alle Register	178
XCHG	Austausch von Operanden	198
XLAT	Laden von Tabellenelementen	198
IN	Port-Eingabe	147
OUT	Port-Ausgabe	175
LEA	Lade effektive Adresse	163
LDS	Lade Pointer über DS	162
LES	Lade Pointer über ES	162
LAHF	Lade Flagregister nach Register AH	161
SAHF	Setze Flagregister aus Register AH	187
PUSHF	Flagregister im Stack ablegen	181
POPF	Flagregister aus Stack holen	179
Zeichenkettenmanipulationsbefehle		
MOVS	Byte-/Worttransfer	172
INS	Zeichenketteneingabe vom Port	149
OUTS	Zeichenkettenausgabe zum Port	176
CMPS	Zeichenkettenvergleich	140
SCAS	Durchsuchen von Zeichenketten	190
LODS	Lade Zeichenkettenelement	167
STOS	Zeichenkettenelement abspeichern	194

Befehl	Kurzcharakteristik	Seite
REP	Wiederholpräfix für Zeichenkettenbefehle	183
REPE	Wiederholpräfix für Zeichenkettenbefehle	183
REPNE	Wiederholpräfix für Zeichenkettenbefehle	183
REPZ	Wiederholpräfix für Zeichenkettenbefehle	183
REPNZ	Wiederholpräfix für Zeichenkettenbefehle	183
Arithmetikbefehle		
ADD	Addition	129
ADC	Addition mit Übertrag (CF)	129
INC	Inkrementieren um 1	148
AAA	ASCII-Korrektur nach BCD-Addition (ungepackt)	127
DAA	Dezimalkorrektur nach BCD-Addition (gepackt)	142
SUB	Subtraktion	189
SBB	Subtraktion mit Übertrag (CF)	189
DEC	Dekrementieren um 1	143
NEG	Negation (Zweierkomplement)	173
CMP	Vergleich	140
AAS	ASCII-Korrektur nach BCD-Subtraktion (ungepackt)	128
DAS	Dezimalkorrektur nach BCD-Subtraktion (gepackt)	142
MUL	Vorzeichenlose Multiplikation	172
IMUL	Vorzeichenbehaftete Multiplikation	146
AAM	ASCII-Korrektur nach BCD-Multiplikation (ungepackt)	128
DIV	Vorzeichenlose Division	143
IDIV	Vorzeichenbehaftete Division	146
AAD	ASCII-Korrektur vor BCD-Division (ungepackt)	127
CBW	Konvertierung Byte in Wort	137
CWD	Konvertierung Wort in Doppelwort	141
Logikbefehle		
NOT	Invertierung (Einerkomplement)	174
AND	Logisches UND	130
OR	Logisches ODER	175
XOR	Exklusives ODER	199
TEST	Test durch UND (nur Flagbeeinflussung)	196
SAL/SHL	Linksverschiebung arithmetisch/logisch	188
SAR	Rechtsverschiebung arithmetisch	188
SHR	Rechtsverschiebung logisch	188
RCL	Rotieren links durch Flag CF	182
RCR	Rotieren rechts durch Flag CF	182
ROL	Rotieren links	182
ROR	Rotieren rechts	182

Befehl	Kurzcharakteristik	Seite
Programmsteuerbefehle		
JA/JNBE	relativer Sprung bedingt (above, not below equal)	159
JAE/JNB	relativer Sprung bedingt (above equal, not below)	159
JB/JNAE	relativer Sprung bedingt (below, not above equal)	159
JBE/JNA	relativer Sprung bedingt (below equal, not above)	159
JC	relativer Sprung bedingt (carry)	159
JE/JZ	relativer Sprung bedingt (equal, zero)	159
JG/JNLE	relativer Sprung bedingt (greater, not less equal)	159
JGE/JNL	relativer Sprung bedingt (greater equal, not less)	159
JL/JNGE	relativer Sprung bedingt (less, not greater equal)	159
JLE/JNG	relativer Sprung bedingt (less equal, not greater)	159
JNC	relativer Sprung bedingt (no carry)	159
JNE/JNZ	relativer Sprung bedingt (not equal, not zero)	159
JNO	relativer Sprung bedingt (no overflow)	159
JNP/JPO	relativer Sprung bedingt (no parity, parity odd)	159
JNS	relativer Sprung bedingt (no sign)	159
JO	relativer Sprung bedingt (overflow)	159
JP/JPE	relativer Sprung bedingt (parity, parity even)	159
JS	relativer Sprung bedingt (sign)	159
JMP	Sprungbefehl	155
CALL	Unterprogrammaufruf	133
RET	Rückkehr aus einem Unterprogramm	185
LOOP	DEC CX; rel. Sprung falls CX ungleich null	168
LOOPE	DEC CX; rel. Sprung falls CX ungleich null und ZF=1	168
LOOPNE	DEC CX; rel. Sprung falls CX ungleich null und ZF=0	168
LOOPZ	DEC CX; rel. Sprung falls CX ungleich null und ZF=1	168
LOOPNZ	DEC CX; rel. Sprung falls CX ungleich null und ZF=0	168
JCXZ	rel. Sprung (Register CX=0)	159
INT	Interrupt	150
INTO	Interrupt 4, falls OF=1	150
IRET	Rückkehr vom Interrupt	152

13.3. Befehlsbeschreibung

AAA Korrektur nach Addition ungepackter BCD-Ziffern **AAA**

Opkode	Befehl	Takte	Beschreibung
37	AAA	3	if ((AL) & 0FH) > 9 or (AF) = 1) then (AL) = (AL) + 6 (AH) = (AH) + 1 (AF) = 1 (CF) = (AF) (AL) = (AL) & 0FH

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	U	-	-	-	U	U	M	U	M

Operation:

Der Befehl korrigiert das Ergebnis einer vorangegangenen Addition zweier ungepackter BCD-Zahlen im Register AL so, daß in AL wieder eine gültige ungepackte BCD-Zahl entsteht.
Entstand bei der Addition in den niederwertigen vier Bitpositionen des Registers AL ein dezimaler Übertrag (AL größer als 9 oder AF=1), wird Register AH um den Wert 1 erhöht (Übertrag) und die Ziffer in AL dezimal korrigiert. Die Flags AF und CF werden gesetzt. Die vier höherwertigen Bitpositionen des Registers AL sind immer gleich null. Trat kein dezimaler Übertrag auf, bleibt AX unverändert, und die Flags CF und AF sind rückgesetzt.
Soll der korrigierte Wert in AL in ein ASCII-Zeichen konvertiert werden, ist nach dem Befehl AAA der Befehl OR AL,30H zu benutzen.

Keine Exceptions.

AAD Korrektur vor Division ungepackter BCD-Ziffern **AAD**

Opkode	Befehl	Takte	Beschreibung
D5 0A	AAD	14	AL := (AH * 10) + AL AH := 0

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	U	-	-	-	M	M	U	M	U

Operation:

Der Befehl bereitet zwei ungepackte BCD-Ziffern (oder ASCII-Zeichen) einer Zahl, deren höherwertige Ziffer in AH und deren niederwertige Ziffer in AL steht, für eine Division vor. Von beiden Ziffern werden jeweils nur die niederwertigen 4 Bits benutzt.
AX enthält anschließend das binäre Äquivalent der ungepackten, aus zwei Ziffern bestehenden BCD-Zahl. Ein nachfolgender Divisionsbefehl liefert ohne nochmalige Korrektur das richtige Ergebnis.

Keine Exceptions.

AAM Korrektur nach Multiplikation ungepackter BCD-Ziffern **AAM**

Opkode	Befehl	Takte	Beschreibung
D4 0A	AAM	16	AH := AL / 10 AL := AL % 10

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	U	-	-	-	M	M	U	M	U

Operation:

Der Befehl AAM korrigiert das Ergebnis einer vorangegangenen Multiplikation zweier ungepackter BCD-Ziffern im Register AL so, daß in AX wieder zwei ungepackte BCD-Ziffern entstehen. Dazu wird das Produkt in AL durch 10 dividiert. Der Quotient wird in AH abgelegt (höherwertige Ziffer), der Rest der Division verbleibt in AL (niederwertige Ziffer).
In beiden Registern werden jeweils nur die niederwertigen 4 Bits benutzt.

Keine Exceptions.

AAS Korrektur nach Subtraktion ungepackter BCD-Ziffern **AAS**

Opkode	Befehl	Takte	Beschreibung
3F	AAS	3	if ((AL) & 0FH) > 9 or (AF) = 1 then (AL) = (AL) - 6 (AH) = (AH) - 1 (AF) = 1 (CF) = (AF) (AL) = (AL) & 0FH

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	U	-	-	-	U	U	M	U	M

Operation:

Der Befehl korrigiert das Ergebnis einer vorangegangenen Subtraktion zweier ungepackter BCD-Zahlen, das sich im Register AL befinden muß. In AL entsteht anschließend wieder eine ungepackte BCD-Ziffer. Die Operanden für die vorausgehende Subtraktion sollten dabei jedoch im Zahlenbereich von 0 bis 9 liegen.
Ist der Wert in AL in den niederwertigen vier Bitpositionen kleiner als 9 und ist das Flag AF rückgesetzt (keine Über- oder Unterschreitung des Zehners), erfolgt keine Korrektur. Das Register AH bleibt unverändert, und die Flags CF und AF sind rückgesetzt. Anderenfalls wird das Register AL dezimal korrigiert und AH um den Wert 1 dekre-

mentiert. Die Flags AF und CF werden gesetzt. Die vier höherwertigen Bitpositionen von AL sind immer gleich null.
Der korrigierte Wert kann durch einen nachfolgenden Befehl OR AL,30H in ein ASCII-Zeichen konvertiert werden.

Keine Exceptions.

ADC/ADD Addition von Integerzahlen **ADC/ADD**

Opkode	Befehl	Takte	Beschreibung
10 /r	ADC eb,rb	2,mem=7	eb = eb + rb + CF
11 /r	ADC ew,rw	2,mem=7	ew = ew + rw + CF
12 /r	ADC rb,eb	2,mem=7	rb = rb + eb + CF
13 /r	ADC rw,ew	2,mem=7	rw = rw + ew + CF
14 db	ADC AL,db	3	AL = AL + db + CF
15 dw	ADC AX,dw	3	AX = AX + dw + CF
80 /2 db	ADC eb,db	3,mem=7	eb = eb + db + CF
81 /2 dw	ADC ew,dw	3,mem=7	ew = ew + dw + CF
83 /2 db	ADC ew,db	3,mem=7	ew = ew + db + CF
00 /r	ADD eb,rb	2,mem=7	eb = eb + rb
01 /r	ADD ew,rw	2,mem=7	ew = ew + rw
02 /r	ADD rb,eb	2,mem=7	rb rb + eb
03 /r	ADD rw,ew	2,mem=7	rw = rw + ew
04 db	ADD AL,db	3	AL = AL + db
05 dw	ADD AX,dw	3	AX = AX + dw
80 /0 db	ADD eb,db	3,mem=7	eb = eb + db
81 /0 dw	ADD ew,dw	3,mem=7	ew = ew + dw
83 /0 db	ADD ew,db	3,mem=7	ew = ew + db

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	M	-	-	-	M	M	M	M	M

Operation:
Mit Hilfe der Befehle ADC und ADD können zwei Integerzahlen addiert werden. Der Befehl ADC addiert einen vorhandenen Übertrag (angegeben durch den Wert des Flags CF vor der Befehlsausführung) zum Ergebnis. Das Ergebnis wird im ersten Operanden abgelegt. ADC wird meist bei der Addition von Integeroperanden benutzt, die aus mehreren Worten bestehen.
Direkt angegebene Byteoperanden, die mit Wortoperanden addiert werden sollen, werden vom Mikroprozessor intern vor der Addition vorzeichenbehaftet auf Wortgröße erweitert.

Exceptions im PVA-Mode:
#GP(0) Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13 Der Wortoperand besitzt ein Offset von OFFFFH.

AND — Logisches UND — AND

Opkode	Befehl	Takte	Beschreibung
20 /r	AND eb,rb	2,mem=7	eb := eb UND rb
21 /r	AND ew,rw	2,mem=7	ew := ew UND rw
22 /r	AND rb,eb	2,mem=7	rb := rb UND eb
23 /r	AND rw,ew	2,mem=7	rw := rw UND ew
24 db	AND AL,db	3	AL := AL UND db
25 dw	AND AX,dw	3	AX := AX UND dw
80 /4 db	AND eb,db	3,mem=7	eb := eb UND db
81 /4 dw	AND ew,dw	3,mem=7	ew := ew UND dw

Flag:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	0	-	-	-	M	M	U	M	0

Operation:
Der Befehl realisiert eine bitweise logische UND-Verknüpfung der Operanden. Enthalten korrespondierende Bitpositionen beider Operanden den Wert 1, wird das Resultatbit ebenfalls auf den Wert 1 gesetzt. In jedem anderen Fall erhält das Resultatbit den Wert 0. Das Resultat wird im Zieloperanden abgelegt.

Exceptions im PVA-Mode:
#GP(0) Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13 Der Wortoperand besitzt ein Offset von OFFFFH.

ARPL — Korrektur des RPL-Feldes eines Selektors — ARPL

Opkode	Befehl	Takte	Beschreibung

63 /r	ARPL ew,rw	10,mem=11	if (RPL(ew) < RPL(rw)) then
			RPL(ew) := RPL(rw)
			ZF := 1
			else
			ZF := 0

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	M	-	-	-

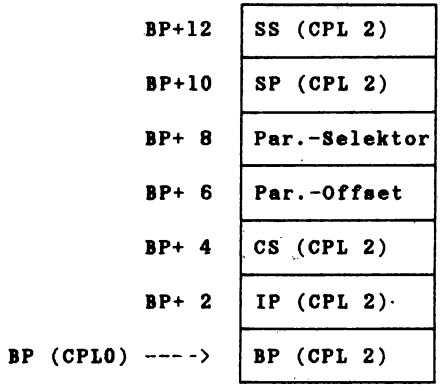
Operation:

Dieser Befehl ist nur für die Nutzung in Betriebssystemen gedacht. Er unterstützt die Hardwareschutzmechanismen des Mikroprozessors 80286. Seine Anwendung ist in Unterprogrammen sinnvoll, wenn gesichert werden muß, daß ein Pointeroperand, der von einem niedriger privilegierten Programm als Parameter an das Unterprogramm übergeben wurde, keine unerlaubten Manipulationen mit höher privilegierten Daten ermöglicht.

Das RPL-Feld (Bit 0 und 1) des ersten Wortoperanden (Selektor des übergebenen Parameters) wird mit dem RPL-Feld des zweiten Operanden (i.allg. der Selektor des aufrufenden Programms) verglichen. Ist die Privilegierungsstufe des ersten Operanden höher (numerisch kleiner) als die des zweiten, wird die Privilegierungsstufe des ersten Operanden auf die Privilegierungsstufe des zweiten Operanden herabgesetzt (numerisch erhöht) und das Flag ZF gesetzt. Anderenfalls bleiben beide RPL-Felder unverändert, und das Flag ZF ist null.

Beispiel:

Ein Unterprogramm in der Privilegierungsstufe 0 wird mit einem Pointerparameter von einem Programm der Privilegierungsstufe 2 aufgerufen. Im Stack des aufgerufenen Unterprogramms (Privilegierungsstufe 0) befindet sich folgende Datenstruktur:



Beinhaltet jetzt das RPL-Feld des als Parameter übergebenen Segmentselektors (BP+8) den Wert 01 (Privilegierungsstufe 1), könnte im Unterprogramm über diesen Pointer auf Daten der Privilegierungsstufe 1 unerlaubt zugegriffen werden. Folgende Befehlsfolge im Unterprogramm verhindert diesen unerlaubten Zugriff:

```
MOV     AX, [BP+4]  ;AX enthält den Selektor des aufrufenden
                    ;Programms aus der Rückkehradresse und damit
                    ;die Privilegierung des aufrufenden Programms
ARPL    [BP+8], AX  ;Korrektur des Parameterselektor-RPL-Feldes
```

Das RPL-Feld des als Parameter übergebenen Pointerselektors wird von 01 auf 10 erhöht (Verringerung der Priorität auf Privilegierungsstufe 2) und das Flag ZF wird gesetzt.

Exceptions im PVA-Mode:

#GP(0)	Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 6 - Der Befehl ARPL ist im RA-Mode nicht zulässig.

BOUND — Grenzwerttest — BOUND

Opkode	Befehl	Takte	Beschreibung
62 /r	BOUND rw,md	noj=13	if (rw < (md) or rw > (md+2)) then Interrupt 5

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Mit dem Befehl BOUND wird geprüft, ob der vorzeichenbehaftete Inhalt des Registers innerhalb angegebener Grenzwerte liegt (z.B. ein Feldindex innerhalb eines Feldes). Die Grenzwerte werden als zwei aufeinanderfolgende Worte (Reihenfolge: untere Grenze, obere Grenze) unter der durch den zweiten Operanden bezeichneten Speicheradresse vorgegeben. Die Grenzwerte selbst sind Bestandteil des vorgegebenen Intervalls. Liegt der Wert des Registers außerhalb des zulässigen Intervalls, wird Interrupt 5 generiert. In der zugehörigen Behandlungsroutine kann die Fehlerauswertung erfolgen.
Es ist günstig, die Grenzwerte im Speicher vor dem Feld anzuordnen. Sie sind dann über das fest vorgegebene Offset von -4 bezogen auf die Adresse des Feldes erreichbar.

Exceptions im PVA-Mode:

Interrupt 5	Es wurde ein Grenzwert überschritten.
#GP(0)	Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.
#UD	Der zweite Operand bezeichnet ein Register.

Exceptions im RA-Mode:

Interrupt 5	Es wurde ein Grenzwert überschritten.
Interrupt 13	Die Speicheradresse (zweiter Operand) besitzt ein Offset von OFFFDH oder größer.
Interrupt 6	Der zweite Operand bezeichnet ein Register.

CALL Aufruf eines Unterprogramms **CALL**

Opkode	Befehl	Takte	Beschreibung
E8 cw	CALL cw	7	CALL NEAR (Offset relativ zum folgenden Befehl)
FF /2	CALL ew	7,mem=11	CALL NEAR indirekt (Offset absolut)
9A cd	CALL cd	13,pm=26	CALL FAR (4-Byte-Direktadresse)
9A cd	CALL cd	41	CALL FAR über CALL-Gate, DPL=CPL
9A cd	CALL cd	82	CALL FAR über CALL-Gate, DPL<CPL, o. Parameter
9A cd	CALL cd	86+4*n	CALL FAR über CALL-Gate, DPL<CPL, n Parameter
9A cd	CALL cd	177	CALL FAR über Task-State-Segment (TSS)
9A cd	CALL cd	182	CALL FAR über Task-Gate
FF /3	CALL ed	16,mem=29	CALL FAR indirekt
FF /3	CALL ed	44	CALL FAR über CALL-Gate, DPL=CPL
FF /3	CALL ed	83	CALL FAR über CALL-Gate, DPL<CPL, o. Parameter
FF /3	CALL ed	90+4*n	CALL FAR über CALL-Gate, DPL<CPL, n Parameter
FF /3	CALL ed	180	CALL FAR über Task-State-Segment (TSS)
FF /3	CALL ed	185	CALL FAR über Task-Gate

Flags:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
-	-	-	-	-	-	-	-	-

Ausnahme: Erfolgt ein Taskwechsel, können OF, SF, ZF, AF, PF und CF modifiziert werden.

Operation:

Der Befehl CALL ermöglicht die Programmausführung eines Unterprogramms. Vor der Verzweigung wird die Rückkehradresse (die Adresse des Befehls, der dem Befehl CALL folgt) im Stack abgelegt. Unterproramme sollte i. allg. mit einem Befehl RET (Rückkehr aus dem Unterprogramm) abgeschlossen werden. Dieser Befehl lädt die im Stack gespeicherte Rückkehradresse in den Instruktionspointer zurück und führt die Programmabarbeitung fort.
Bei Unterprogrammaufrufen werden folgende Varianten unterschieden:

- Unterprogrammaufruf im aktuellen Programmkodesegment:
  + CALL NEAR : Distanzwert 16 Bit (innerhalb des aktuellen 64-KByte-Segments
- Unterprogrammaufruf in anderen Programmkodesegmenten (RA-Mode):
  + CALL FAR : vollständige Direktadresse
- Unterprogrammaufruf in anderen Programmkodesegmenten (PVA-Mode):
  + CALL FAR : vollständige Adresse in einem Kodesegment
  + CALL FAR : vollständige Adresse eines CALL-Gates
  + CALL FAR vollständige Adresse eines Task-Gates
  + CALL FAR vollständige Adresse eines Task-State-Segments.

- CALL NEAR

In der ersten Form (CALL NEAR) wird das aktuelle Programmkodesegment nicht verlassen, d.h., das Register CS bleibt unverändert. Im Stack wird nur die Offsetkomponente der Rückkehradresse abgelegt. Bei direkter Angabe des Sprungziels (Form CALL cw) wird der Operand zum

Offset des dem CALL-Befehl folgenden Befehls addiert (modulo 65536) und Register IP auf den ermittelten Wert gesetzt.
Ist die Offsetkomponente des Sprungziels indirekt angegeben (CALL ew), wird der Inhalt der effektiven Adresse als absolutes Offset gewertet (bezogen auf den Anfang des 64-KByte-Programmkodesegments). Der Aufruf von Unterprogrammen innerhalb des aktuellen Programmkodesegments funktioniert in beiden Betriebsarten des Mikroprozessors 80286 gleich.

Für den Aufruf von Unterprogrammen, die sich in anderen Programmkodesegmenten befinden, stehen verschiedene Varianten des Befehls (CALL FAR) zur Verfügung, die im folgenden erläutert werden. Bei diesen Varianten kann der Operand direkt oder indirekt angegeben werden.

- CALL FAR über eine Direktadresse (4-Byte-Pointer)
  Der Prozessor überprüft die Zugriffsrechte auf das Programmkodesegment (Accessbyte im Deskriptor), rettet die vollständige Rückkehradresse (CS:IP) im Stack und arbeitet das Unterprogramm ab. Der Unterprogrammaufruf über eine Direktadresse ist in beiden Betriebsarten des 80286 ausführbar, jedoch erfolgt im RA-Mode kein Test der Zugriffsberechtigung.

- CALL FAR über ein CALL-Gate
  Der im Unterprogrammaufruf angegebene Selektor zeigt auf einen CALL-Gate-Deskriptor in der GDT oder der LDT (die Offsetkomponente muß vorhanden sein, wird aber ignoriert). Im PVA-Mode prüft der Prozessor die Zugriffsrechte des aufrufenden Prozesses auf den CALL-Gate-Deskriptor selbst und auf den im CALL-Gate spezifizierten Programmkodeselektor. Ist das spezifizierte Unterprogramm höher privilegiert als das aufrufende Programm (DPL des Selektors ist kleiner als CPL), wird aus dem Task-State-Segment der aktuellen Task der zu der Privilegierungsstufe des aufgerufenen Unterprogramms gehörende Stackpointer geladen. Der Stackpointer des aufrufenden Programms wird im neuen Stack gerettet. Anschließend wird die im CALL-Gate-Deskriptor angegebene Anzahl von Parameterworten (maximal 31 Worte) in den Stack des Unterprogramms kopiert, bevor es ausgeführt wird.

- CALL FAR mit Taskwechsel über ein Task-Gate
  Im Unterprogrammaufruf ist ein Selektor spezifiziert, der entweder in der GDT oder der LDT auf einen Task-Gate-Deskriptor zeigt. Der in diesem Task-Gate-Deskriptor enthaltene Selektor zeigt seinerseits auf ein Task-State-Segment in der GDT. Bei Ausführung des CALL-Befehls wird der Kontext der aktuellen Task in dem Task-State-Segment gerettet, das dieser aufrufenden Task zugeordnet ist (zugreifbar über den aktuellen Inhalt des Taskregister TR).
  Anschließend werden die Register des 80286 mit den Werten geladen, die in dem Task-State-Segment gespeichert waren, das im Task-Gate spezifiziert ist. Der Selektor der aufrufenden Task wird aus dem Taskregister in das Back-Link-Feld der aufgerufenen Task geladen, um die Rückkehr zu ermöglichen. Das Taskregister erhält automatisch den Selektor des neuen TSS. Das Flag NT, das eine Taskverschachtelung anzeigt, wird im Flagregister der neuen Task gesetzt. Die Ausführung der aufgerufenen Task beginnt mit dem Befehl, der in ihrer letzten

```
    aktiven Phase nicht mehr ausgeführt wurde.

- CALL FAR mit Taskwechsel über ein Task-State-Segment
    Der  Taskwechsel erfolgt wie bei einem Unterprogrammaufruf  über  ein
    Task-Gate.  Der angegebene Selektor muß jedoch ein Task-State-Segment
    in der GDT spezifizieren.
```

```
Exceptions im PVA-Mode:

- if Operand über indirekte Adressierung then
   - Zugriff auf effektive Doppelwortadresse ist erlaubt else #GP(0)
- CS-Selektoroperand ist ungleich null else #GP(0)
- CS-Selektoroperand  liegt  innerhalb der Grenzen  der  Deskriptortabelle
  else #GP(Selektor)
- Test Accessbyte des durch den Selektor spezifizierten  Deskriptors:

   CALL zu Programmkodesegment (conforming)

   - DPL <= CPL else #GP(CS-Selektor)
   - Segment ist präsent else #NP(CS-Selektor)
   - im  Stack  ist  genügend Platz für  die  Ablage  der  Rückkehradresse
     verfügbar else #SS(0)
   - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
   - CS := Programmkodeselektor
   - CS-Cache := Segmentdeskriptor
   - IP := Offset des Operanden

   CALL zu Programmkodesegment (non-conforming)

   - RPL <= CPL else #GP(CS-Selektor)
   - DPL = CPL else #GP(CS-Selektor)
   - Segment ist präsent else #NP(CS-Selektor)
   - im  Stack  ist  genügend Platz für  die  Ablage  der  Rückkehradresse
     verfügbar else #SS(0)
   - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
   - CS := Programmkodeselektor
   - CS-Cache := Segmentdeskriptor
   - RPL-Feld des CS-Registers := CPL
   - IP := Offset des Operanden

   CALL zu CALL-Gate

   - DPL des CALL-Gates >= CPL else #GP(CALL-Gateselektor)
   - DPL des CALL-Gates >= RPL else #GP(CALL-Gateselektor)
   - CALL-Gate ist präsent else #NP(CALL-Gateselektor)
   - Test CS-Selektor im CALL-Gatedeskriptor:
      - Selektor ist ungleich null else #GP(0)
      - Selektor liegt innerhalb der Grenzen der Deskriptortabelle
        else #GP(CS-Selektor)
      - Deskriptor  (Accessbyte) beschreibt ein Kodesegment  else  #GP(CS-
        Selektor)
      - DPL des selektierten Deskriptors <= CPL else #GP(CS-Selektor)
      - if Kodesegment ist non-conforming und DPL < CPL then
```

```
        CALL-Gate zu höherer Privilegierung

        - SS-Selektor für neue Privilegstufe := TSS
        - Test Selektor und Deskriptor für neues SS:
           - Selektor ist ungleich null else #TS(0)
           - Selektor liegt innerhalb der Grenzen der Deskriptortabelle
             else #TS(SS-Selektor)
           - RPL des Selektors = DPL des Kodesegments else  #TS(SS-Selek-
             tor)
           - DPL  des Stacksegments = DPL des Kodesegments  else  #TS(SS-
             Selektor)
           - Deskriptor (Accessbyte) beschreibt ein beschreibbares Daten-
             segment else #TS(SS-Selektor)
           - Segment ist präsent else #SS(SS-Selektor)
        - im neuen Stack ist Platz für Parameter plus 8 Bytes else #SS(0)
        - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
        - SS:SP := TSS
        - CS:IP := CALL-Gate
        - CS- und SS-Cache := Deskriptoren
        - PUSH alten Stackpointer (4 Byte) in den neuen Stack
        - Wortanzahl aus CALL-Gatedeskriptor wird mit 01FH maskiert
        - Parameterübertragung aus dem alten in den neuen Stack
        - PUSH Rückkehradresse in den neuen Stack
        - CPL := DPL des Stacksegments
        - RPL-Feld des CS-Registers := CPL

    else

        CALL-GATE zu gleicher Privilegierung

        - im Stack ist Platz für 4 Bytes (Rückkehradresse) else #SS(0)
        - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
        - CS:IP := CALL-Gate
        - PUSH Rückkehradresse in den Stack
        - CS-Cache := Deskriptor
        - RPL-Feld des CS-Registers := CPL

CALL zu Task-Gate

- DPL des Task-Gates >= CPL else #GP(Task-Gateselektor)
- DPL des Task-Gates >= RPL else #GP(Task-Gateselektor)
- Task-Gate ist präsent else #NP(Task-Gateselektor)
- Test TSS-Selektor im Task-Gatedeskriptor:
   - TSS-Selektor zeigt in GDT else #GP(TSS-Selektor)
   - TSS-Selektor  liegt  innerhalb  der  GDT-Grenzen  else  #GP(TSS-
     Selektor)
   - Deskriptor  (Accessbyte)  beschreibt  ein  verfügbares  TSS  else
     #GP(TSS-Selektor)
   - TSS ist präsent else #NP(TSS-Selektor)
- Taskwechsel mit Verschachtelung zu TSS
- IP liegt innerhalb des Kodesegmentlimits else #GP(0)
```

CALL zu Task-State-Segment

- DPL des TSS >= CPL else #GP(TSS-Selektor)
- DPL des TSS >= RPL else #GP(TSS-Selektor)
- Deskriptor (Accessbyte) beschreibt ein verfügbares TSS else #GP(TSS-Selektor)
- TSS ist präsent else #NP(TSS-Selektor)
- Taskwechsel mit Verschachtelung zu TSS
- IP liegt innerhalb des Kodesegmentlimits else #GP(0)

ELSE #GP(CS-Selektor)

Exceptions im RA-Mode:
Interrupt 13 Der Wortoperand besitzt ein Offset von OFFFFH (CALL NEAR indirekt) byw. OFFFDH (CALL FAR indirekt).

CBW — Vorzeichenbehaftete Konvertierung Byte in Wort — CBW

Opkode	Befehl	Takte	Beschreibung
98	CBW	2	AX := AL (Vorzeichenerweiterung)

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:
Der vorzeichenbehaftete Wert im Byteregister AL wird in einen vorzeichenbehafteten Wert im Wortregister AX umgewandelt. Das Vorzeichen (Bit 7 des Wertes im Register AL) wird in jede Bitposition des Registers AH übertragen (Abshcn. 11. Datenformate).

Keine Exceptions.

CLC — Löschen des Flags CF — CLC

Opkode	Befehl	Takte	Beschreibung
F8	CLC	2	CF := 0

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	0

Operation:
Der Befehl CLC setzt das Flag CF (carry flag) im Flagregister zurück. Er wird z.B. zur Schaffung definierter Ausgangszustände vor Rotations- oder Verschiebebefehlen genutzt (vgl. Befehle STC, CMC).

Keine Exceptions.

CLD Löschen des Flags DF **CLD**

Opkode	Befehl	Takte	Beschreibung
FC	CLD	2	DF := 0

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	0	-	-	-	-	-	-	-

Operation:

Bei Ausführung des Befehls CLD wird das Flag DF (direction flag) im Flagregister rückgesetzt. Das Flag DF bestimmt die Richtung für das automatische Weitersetzen der Pointer (Indexregister SI bzw. DI) bei der Ausführung der Zeichenkettenbefehle der 80286-CPU. Nach Ausführung des Befehls CLD (DF=0) werden Zeichenketten in steigender Adreßrichtung verarbeitet (vgl. Befehl STD).

Keine Exceptions.

CLI Löschen des Interruptflags **CLI**

Opkode	Befehl	Takte	Beschreibung
FA	CLI	3	IF := 0 ;(Vor.: CPL <= IOPL)

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	0	-	-	-	-	-	-

Operation:

Das Interruptflag IF im Flagregister wird rückgesetzt, wenn die aktuelle Privilegierungsstufe mindestens der Stufe entspricht, die durch die Bitpositionen IOPL im Flagregister eingestellt ist. Externe Interruptanforderungen werden nach der Bofehlsabarbeitung so lange nicht behandelt, bis das Flag IF wieder gesetzt ist (z.B. durch den Befehl STI).

Exceptions im PVA-Mode:

#GP(0) Die aktuelle Privilegierungsstufe ist größer (niedrigerer Vorrang) als die Stufe, die durch die Bitpositionen IOPL eingestellt ist.

Exceptions im RA-Mode: keine

CLTS Löschen des TS-Bit im Maschinenstatuswort **CLTS**

Opkode	Befehl	Takte	Beschreibung
0F 06	CLTS	2	TS := 0 ;(Vor.: CPL = 0)

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Das Bit TS (task switched flag) im Maschinenstatuswort wird rückgesetzt. Dieses Bit, das bei jedem Taskwechsel automatisch gesetzt wird, dient zur Steuerung des Numerikcoprozessors.
Ist das Bit MP (math present flag) im Maschinenestatuswort gesetzt und tritt ein Taskwechsel während der Ausführung der Befehle WAIT oder ESC auf, wird ein Trap erzeugt. Wurde in der "alten" Task der Numerikcoprozessor benutzt und ist zum Zeitpunkt des Taskwechsels die Operation im Coprozessor noch nicht beendet, muß der Kontext des Coprozessors gerettet werden, bevor in der neuen Task der Coprozessor erneut benutzt werden kann. Die Fehlerroutine muß den Kontext des Coprozessors sichern und das TS-Bit rücksetzen oder die "neue" Task, die den Coprozessor benutzen will, in eine Warteschlange einreihen, bis die aktuelle Befehlsausführung des Coprozessors beendet ist.
Der privilegierte Befehl CLTS ist für Betriebssystemprogramme gedacht. Er kann nur in der Privilegierungsstufe 0 (höchste Privilegierung) fehlerfrei abgearbeitet werden.

Exceptions im PVA-Mode:
#GP(0) CLTS wurde in einer Privilegierungsstufe ungleich 0 ausgeführt.

Exceptions im RA-Mode: keine

CMC Komplementbildung des Flags CF **CMC**

Opkode	Befehl	Takte	Beschreibung
F5	CMC	2	CF = /CF

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	M

Operation:

Der Wert des Flags CF (carry flag) im Flagregister wird umgekehrt (vgl. Befehle CLC, STC).

Keine Exceptions.

CMP Vergleich zweier Operanden **CMP**

Opkode	Befehl	Takte	Beschreibung
38 /r	CMP eb,rb	2,mem=7	eb - rb ?
39 /r	CMP ew,rw	2,mem=7	ew - rw ?
3A /r	CMP rb,eb	2,mem=6	rb - eb ?
3B /r	CMP rw,ew	2,mem=6	rw - ew ?
3C db	CMP AL,db	3	AL - db ?
3D dw	CMP AX,dw	3	AX - dw ?
80 /7 db	CMP eb,db	3,mem=6	eb - db ?
83 /7 db	CMP ew,db	3,mem=6	ew - db ?
81 /7 dw	CMP ew,dw	3,mem=6	ew - dw ?

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	M	-	-	-	M	M	M	M	M

Operation:

Der Vergleich wird durch Subtraktion des zweiten Operanden vom ersten Operanden realisiert. Das Ergebnis wird jedoch nicht im Zieloperanden übergeben, sondern der Befehl beeinflußt nur die Flags, die mit Hilfe nachfolgender bedingter Sprungbefehle für Programmverzweigungen genutzt werden können.

Soll ein Wortoperand mit einem 8-Bit-Direktwert verglichen werden, wird der 8-Bit-Direktwert im Mikroprozessor intern vor dem Vergleich vorzeichenbehaftet zu einem 16-Bit-Wert erweitert.

Exceptions im PVA-Mode:

#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

CMPS Vergleich von Zeichenketten **CMPS**

Opkode	Befehl	Takte	Beschreibung
A6	CMPS mb,mb	8	Bytevergleich ES:[DI] - [SI] ; SI, DI setzen
A7	CMPS mw,mw	8	Wortvergleich ES:[DI] - [SI] ; SI, DI setzen
A6	CMPSB	8	Bytevergleich ES:[DI] - DS:[SI] ; SI, DI setzen
A7	CMPSW	8	Wortvergleich ES:[DI] - DS:[SI] ; SI, DI setzen

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	M	-	-	-	M	M	M	M	M

Operation:

Der Befehl vergleicht zwei Elemente (Bytes, Worte) von im Speicher befindlichen Zeichenketten durch Subtraktion, wobei als Ergebnis nur die oben angegebenen Bitpositionen im Flagregister beeinflußt werden.

Die Operanden bleiben unverändert. Die Elemente der Zeichenkette werden durch die Indexregister SI und DI adressiert.

Der Zieloperand muß immer über das Registerpaar ES:DI adressiert werden. Für den Quelloperanden kann in der Form CMPS ein Segment-Override-Präfix zur Auswahl eines beliebigen Segmentregisters spezifiziert werden. Bei Formen CMPSB und CMPSW ist das Segmentregister DS für den Quelloperanden voreingestellt.
Nach dem Vergleich werden die Register SI und DI in Abhängigkeit vom Flag DF (direction flag) wie folgt verändert:

	DF = 0	DF = 1
Bytevergleich	SI = SI + 1 DI = DI + 1	SI = SI - 1 DI = DI - 1
Wortvergleich	SI = SI + 2 DI = DI + 2	SI = SI - 2 DI = DI - 2

Zum Vergleich ganzer Zeichenketten können den Befehlen CMPS die Präfixe REPE oder REPNE vorangestellt werden. Der Vergleich wird dann abgebrochen, wenn entweder die im Präfix angegebene Bedingung nicht mehr erfüllt ist oder das Register CX, in dem die Anzahl der Wiederholungen vorgegeben wurde, den Wert 0 erreicht hat (siehe Präfix REP).

Exceptions im PVA-Mode:
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13 Der Wortoperand besitzt ein Offset von OFFFFH.

CWD Vorzeichenbehaftete Konvertierung Wort in Doppelwort CWD

Opkode	Befehl	Takte	Beschreibung
99	CWD	2	DX:AX := AX (Vorzeichenerweiterung)

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:
Der vorzeichenbehaftete Wortoperand im Register AX wird in einen vorzeichenbehafteten 32-Bit-Wert im Doppelregister DX:AX umgewandelt. Das Vorzeichen (Bit 15 des Wertes im Register AX) wird in jede Bitposition des Registers DX übertragen (Abschn. 11. Datenformate).

Keine Exceptions.

DAA Dezimalkorrektur gepackter BCD-Zahlen nach Addition **DAA**

Opkode	Befehl	Takte	Beschreibung

27	DAA	3	if ((AL) & 0FH) > 9 or (AF) = 1 then
			(AL) := (AL) + 6
			(AF) := 1
			if ((AL) > 9FH) or (CF) = 1 then
			(AL) := (AL) + 60H
			(CF) := 1

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	U	-	-	-	M	M	M	M	M

Operation:

Der Befehl DAA korrigiert das Ergebnis einer Addition zweier gepackter BCD-Zahlen in AL so, daß AL die Summe als zweistellige Zahl im gepackten BCD-Format enthält.

Keine Exceptions.

DAS Dezimalkorrektur gepackter BCD-Zahlen nach Subtraktion **DAS**

Opkode	Befehl	Takte	Beschreibung

2F	DAS	3	if ((AL) & 0FH) > 9 or (AF) = 1 then
			(AL) := (AL) - 6
			(AF) := 1
			if ((AL) > 9FH) or (CF) = 1 then
			(AL) := (AL) - 60H
			(CF) := 1

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	U	-	-	-	M	M	M	M	M

Operation:

Der Befehl DAS korrigiert das Ergebnis einer Subtraktion zweier gepackter BCD-Zahlen in AL so, daß AL die Differenz als zweistellige Zahl im gepackten BCD-Format enthält.

Keine Exceptions.

DEC Dekrementieren **DEC**

Opkode	Befehl	Takte	Beschreibung
FE /1	DEC eb	2,mem=7	eb = eb - 1
FF /1	DEC ew	2,mem=7	ew = ew - 1
48+ rw	DEC rw	2	rw = rw - 1

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	M	-	-	-	M	M	M	M	-

Operation:

Vom Inhalt des Zieloperanden wird 1 subtrahiert. Das Flag CF wird nicht beeinflußt. Soll das Flag CF beeinflußt werden, ist der Befehl SUB mit dem Direktoperanden 1 zu benutzen.

Exceptions im PVA-Mode:

#GP(0)	Der Operand befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

DIV Vorzeichenlose Division **DIV**

Opkode	Befehl	Takte	Beschreibung
F6 /6	DIV eb	14,mem=17	AL = AX / eb ; AH = AX % eb
F7 /6	DIV ew	22,mem=25	AX = DX:AX / ew ; DX = DX:AX % ew

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	U	-	-	-	U	U	U	U	U

Operation:

Der Inhalt des Akkumulators wird durch den Wert, der unter der im Operanden angegebenen effektiven Adresse steht, dividiert. Dividend und Divisor werden als vorzeichenlose Zahlen interpretiert.

Ist der Quelloperand ein Byte, wird AX durch den Wert des Bytes dividiert; der Quotient wird in AL und der Rest in AH übergeben. Ist der Quelloperand ein Wort, wird das Doppelregister DX:AX (DX enthält den höherwertigen Teil des Dividenden) durch den Wert des Wortes dividiert; der Quotient wird in AX und der Rest in DX übergeben.

Ist der Quotient zu groß, um im Zielregister aufgenommen zu werden (Wert größer 255 bei 8-Bit-Division bzw. größer 65535 bei 16-Bit-Division) oder ist der Divisor null, wird der reservierte Interrupt 0 ausgelöst und der Quotient zu null gesetzt.

Exceptions im PVA-Mode:
Interrupt 0 Der Quotient ist zu groß, oder der Divisor ist 0.
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 0 Der Quotient ist zu groß, oder der Divisor ist 0.
Interrupt 13 Der Wortoperand besitzt ein Offset von OFFFFH.

ENTER Anlegen einer Struktur zur Parameterübergabe im Stack ENTER

Opkode	Befehl	Takte	Beschreibung
C8 dw 00	ENTER dw,0	11	Anlegen eines Stackrahmens (Niveau 0)
C8 dw 01	ENTER dw,1	15	Anlegen eines Stackrahmens (Niveau 1)
C8 dw db	ENTER dw,db	12+4*db	Anlegen eines Stackrahmens (Niveau db)

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl ENTER legt im Stack eine variable Speicherstruktur an, wie sie bei den meisten blockstrukturierten höheren Programmiersprachen zur Parameterübergabe benötigt wird.
Der erste Operand gibt die Größe des dynamisch zugewiesenen Speicherbereichs des Unterprogramms im Stack an. Der zweite Operand bezeichnet die Schachtelungstiefe (das Niveau) des Unterprogramms innerhalb des Quellprogramms in der Hochsprache. Er bestimmt, wie viele Stackrahmenpointer vom übergeordneten Programm in den neuen Stackrahmen kopiert werden. Das Register BP wird als Stackrahmenpointer (frame pointer) benutzt.
Ist der zweite Operand 0, wird BP in den Stack gebracht, BP auf SP gesetzt und SP um den Wert des ersten Operanden erniedrigt (Speicherplatzreservierung für lokale Variablen).

Beispiel:

Ein Unterprogramm mit einem Bereich von 8 Byte für lokale Variablen beginnt mit dem Befehl

```
ENTER 8,0
```

Vor der Beendigung des Unterprogramms mit dem Befehl RET wird der Befehl

```
LEAVE
```

eingefügt, um den Stack wieder zu korrigieren.

Die lokalen Variablen werden im Unterprogramm über negative Offsets zum Register BP adressiert, z.B. über

```
MOV   AX, [BP-6]
```

Der folgende Algorithmus verdeutlicht die Aktionen bei der Ausführung des Befehls ENTER mit größerer Schachtelungstiefe.

```
ENTER size, level

   level := level MOD 32          ; Maskierung fÜr Schachtelungstiefe
   PUSH BP                        ; BP abkellern
   frame_ptr := SP                ; temporärer Wert
   IF (level > 0 ) THEN
       REPEAT (level-1) times:
         BP := BP - 2
         PUSH (BP)                  Wort, auf das BP zeigt, abkellern
       END REPEAT
       PUSH frame_ptr
   END IF
   BP := frame_ptr
   SP := SP - size
```

Exceptions im PVA-Mode:
#SS(0) Der Stackpointer überschreitet das Limit des Stacksegments.

Exceptions im RA-Mode: keine

HLT — Halt — HLT

Opkode	Befehl	Takte	Beschreibung
F4	HLT	2	Halt

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:
Der privilegierte Befehl HLT bringt den Prozessor in den Haltzustand. Die Programmausführung wird nach Empfang und Behandlung eines Interrupts oder nach einem RESET wieder gestartet. Wird die Programmausführung nach einem Interrupt wieder aufgenommen, zeigt der Instruktionpointer CS:IP auf den Befehl, der dem HLT-Befehl folgt.

Exceptions im PVA-Mode:
#GP(0) Die aktuelle Privilegierungsstufe ist größer als 0.

Exceptions im RA-Mode: keine

IDIV — Vorzeichenbehaftete Division — IDIV

Opkode	Befehl	Takte	Beschreibung
F6 /7	IDIV eb	17,mem=20	AL = AX / eb ; AH = AX % eb
F7 /7	IDIV ew	25,mem=28	AX = DX:AX / ew ; DX = DX:AX % ew

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	U	-	-	-	U	U	U	U	U

Operation:

Der Inhalt des Akkumulators wird durch den im Operanden angegebenen Wert dividiert. Beide Zahlen werden als vorzeichenbehaftete Zahlen interpretiert.
Ist der Quelloperand ein Byte, wird AX durch den Wert des Bytes dividiert; der Quotient wird in AL und der Rest in AH übergeben. Ist der Quelloperand ein Wort, wird das Doppelregister DX:AX (DX enthält den höherwertigen Teil) durch den Wert des Wortes dividiert; der Quotient wird in AX und der Rest in DX übergeben.
Ist der Quotient zu groß, um im Zielregister aufgenommen zu werden (außerhalb des Bereiches -127 bis +128 bei 8-Bit-Division bzw. außerhalb -32768 bis +32767 bei 16-Bit-Division) oder ist der Divisor null, wird der reservierte Interrupt 0 ausgelöst und der Quotient zu null gesetzt.
Der Rest hat das gleiche Vorzeichen wie der Dividend.

Exceptions im PVA-Mode:

Interrupt 0	Der Quotient ist zu groß oder der Divisor ist 0.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 0	Der Quotient ist zu groß oder der Divisor ist 0.
Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

IMUL — Vorzeichenbehaftete Multiplikation — IMUL

Opkode	Befehl	Takte	Beschreibung
F6 /5	IMUL eb	13,mem=16	AX = AL * eb
F7 /5	IMUL ew	21,mem=24	DX:AX = AX * ew
6B /r db	IMUL rw,db	21	rw = rw * db
6B /r db	IMUL rw,ew,db	21,mem=24	rw = ew * db
69 /r dw	IMUL rw,ew,dw	21,mem=24	rw = ew * dw

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	M	-	-	-	U	U	U	U	M

Operation:

IMUL mit einem Quelloperanden:

Der Quelloperand wird mit dem Akkumulator vorzeichenbehaftet multipliziert. Das Ergebnis wird im Akkumulator abgelegt. Enthält die höherwertige Hälfte des Ergebnisses in jeder Bitposition den Wert des Vorzeichens der niederwertigen Hälfte des Ergebnisses, werden die Flags CF und OF auf null gesetzt (Produkt liegt noch im Zahlenbereich der Operanden). Anderenfalls haben sie den Wert 1.

Quell-operand	Operation	CF / OF = 0 , wenn gilt:
Byte	AX = AL * eb	AH = 0 oder AH = 0FFH
Wort	DX:AX = AX * ew	DX = 0 oder DX = 0FFFFH

IMUL mit einem Direktoperanden (zwei oder drei Quelloperanden):

Der Operand unter der effektiven Adresse wird mit dem Direktwert multipliziert, wobei die niederwertigen 16 Bit des Ergebnisses in dem durch den ersten Operanden angegebenen Register abgelegt werden.
Kann das Ergebnis vollständig in einem 16-Bit-Register dargestellt werden (-32768 bis 32767), haben die Flags CF und OF den Wert 0, anderenfalls haben sie den Wert 1.

Hinweis:
Die niederwertigen 16 Bit des Ergebnisses einer Multiplikation vorzeichenbehafteter 16-Bit-Werte entsprechen dem Ergebnis einer vorzeichenlosen Multiplikation (vgl. Befehl MUL).

Exceptions im PVA-Mode:

#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von 0FFFFH.

IN — Eingabe von einem Eingabeport — IN

Opkode	Befehl	Takte	Beschreibung
E4 db	IN AL,db	5	AL := (port db)
EC	IN AL,DX	5	AL := (DX)
E5 db	IN AX,db	5	AX := (port db)
ED	IN AX,DX	5	AX := (DX)

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl IN ermöglicht die Dateneingabe von einem Eingabeport, der durch den zweiten Operanden spezifiziert wird. Vom angegebenen Eingabeport wird ein Byte bzw. ein Wort gelesen und im Akkumulator (AL bei Byteeingabe, AX bei Worteingabe) übergeben.
Die Portadresse kann als 8-Bit-Direktwert oder indirekt im Wortregister DX bereitgestellt werden.

Portadresse	Ein-/Ausgabe-Adreßbereich	
Direktwert db	0	0FFH
Register DX	0	0FFFFH

Eingabeports mit einer Breite von 16 Bit werden nur über gerade Portadressen erreicht.
Die Portadressen 0F8H-0FFH sind reserviert.

Exceptions im PVA-Mode:

#GP(0) Die aktuelle Privilegierungsstufe ist höher (niedrigere Priorität) als die Stufe, die durch die Bitpositionen IOPL im Flagregister angegeben ist.

Exceptions im RA-Mode: keine

INC — Inkrementieren — INC

Opkode	Befehl	Takte	Beschreibung
FE /0	INC eb	2,mem=7	eb := eb + 1
FF /0	INC ew	2,mem=7	ew := ew + 1
40+rw	INC rw	2	rw := rw + 1

Flags:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
M	-	-	-	M	M	M	M	-

Operation:

Der Wert 1 wird zum Zieloperanden addiert. Das Flag CF wird nicht beeinflußt. Soll das Flag CF beeinflußt werden, ist der Befehl ADD mit dem Direktoperanden 1 zu benutzen.

Exceptions im PVA-Mode:

#GP(0) Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13 Der Wortoperand besitzt ein Offset von 0FFFFH.

INS Zeichenketteneingabe **INS**

Opkode	Befehl	Takte	Beschreibung
6C	INS eb,DX	5	ES:[DI] := (DX) ; Byteeingabe
6D	INS ew,DX	5	ES:[DI] := (DX) ; Worteingabe
6C	INSB	5	ES:[DI] := (DX) ; Byteeingabe
6D	INSW	5	ES:[DI] := (DX) ; Worteingabe

Flags:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
-	-	-	-	-	-	-	-	-

Operation:

Der Befehl INS ermöglicht die Dateneingabe von einem Eingabeport in einen fortlaufenden Speicherbereich.
Die Portadresse wird im Register DX vorgegeben. Vom Eingabeport wird ein Byte bzw. ein Wort gelesen und im Speicherbereich, der durch das Registerpaar ES:DI angegeben ist, abgelegt.
Nach dem Transfer der Eingabedaten in den Speicher wird das Indexregister DI in Abhängigkeit des Flags DF (direction flag) automatisch beeinflußt:

	DF = 0	DF = 1
Byteeingabe	DI = DI + 1	DI = DI - 1
Worteingabe	DI = DI + 2	DI = DI - 2

Der Speicheroperand kann nur über das Extradatensegmentregister ES adressiert werden (Angabe eines Segment-Override-Präfix ist nicht erlaubt).
Portadressen sind im Bereich 0-0FFFFH zulässig (Ausnahmen: Reservierte Adressen 0F8H-0FFH).
Zur Eingabe von Zeichenketten in Speicherblöcke kann dem Befehl INS der Präfix REP vorangestellt werden. Im Register CX wird dann die Anzahl der Wiederholungen vorgegeben (siehe REP).

Exceptions im PVA-Mode:

#GP(0)	Die aktuelle Privilegierungsstufe ist höher (niedrigere Priorität) als die Stufe, die durch die Bitpositionen IOPL im Flagregister angegeben ist.
#GP(0)	Die Zieladresse befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Bei Worteingabe enthält das Indexregister DI einen Offsetwert von 0FFFFH.

INT/INTO Aufruf einer Interruptprozedur **INT/INTO**

Opkode	Befehl	Takte	Beschreibung
CC	INT 3	23	Interrupt 3 (Debuggertrap, nur RA-Mode)
CC	INT 3	40	Interrupt 3, PVA-Mode, gleiche Privilegstufe
CC	INT 3	78	Interrupt 3, PVA-Mode, höhere Privilegstufe
CC	INT 3	167	Interrupt 3, PVA-Mode, über Task-Gate
CD db	INT db	23	Interrupt n, (nur RA-Mode)
CD db	INT db	40	Interrupt n, PVA-Mode, gleiche Privilegstufe
CD db	INT db	78	Interrupt n, PVA-Mode, höhere Privilegstufe
CD db	INT db	167	Interrupt n, PVA-Mode, über Task-Gate
CE	INTO	24,noj=3	Interrupt 4, falls OF=1

Flags:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
-	-	0	0	-	-	-	-	-

Wurde ein Taskwechsel ausgeführt, sind alle Flags verändert.
Das Flag TF ist immer null, wenn kein Taskwechsel eintrat.
Im RA-Mode wird das Interruptflag IF zu null gesetzt.
Im PVA-Mode wird IF rückgesetzt, wenn sich der Interrupt auf ein Interrupt-Gate bezog.

Operation:

Der Befehl INT bewirkt die Ausführung einer Interruptbehandlungsroutine. Der Direktoparand (ein Wert 0 ... 255) bzw. der implizite Wert 3 stellt einen Index (entspricht einem Interruptvektor) dar, über den in Abhängigkeit von der Betriebsart des 80286 aus der Interruptdeskriptortabelle IDT (im PVA-Mode) bzw. der Interrupttabelle (im RA-Mode) die notwendigen Informationen zum Starten einer Interruptbehandlungsroutine entnommen werden.
Im PVA-Mode muß der ausgewählte Interruptdeskriptor auf ein Interrupt-Gate, ein Trap-Gate oder ein Task-Gate zeigen. Das Flag TF (trap flag) wird immer rückgesetzt. Die ersten 32 Interruptvektoren sind reserviert (Abschn. 9.3. Reservierte Interruptvektoren) und sollten im Befehl nicht spezifiziert werden.

Der Befehl INTO (Interrupt on Overflow) generiert einen Interrupt mit dem Vektor 4, sofern das Flag OF im Flagregister gesetzt ist. Ist OF=0, wird der nächstfolgende Befehl abgearbeitet.

Die Aktionen bei der Ausführung des Befehls INT sind für beide Betriebsarten des 80286 ausführlich im Abschnitt 9 Interruptsystem erläutert.

Exceptions im PVA-Mode:

- Interruptvektor liegt innerhalb der IDT-Grenzen else #GP(Vektornummer*8+2+EXT)
- Deskriptor (Accessbyte) beschreibt ein Interrupt-Gate, Trap-Gate oder Task-Gate else #GP(Vektornummer*8+2+EXT)
- if INT-Befehl then
  - DPL des Gate-Deskriptors >= CPL else #GP(Vektornummer*8+2+EXT)

```
- Gate ist präsent else #GP(Vektornummer*8+2+EXT)
- if Trap-Gate oder Interrupt-Gate then
  - Test CS-Selektor und Deskriptor im Gate-Deskriptor:
     - Selektor ist ungleich null else #GP(EXT)
     - Selektor liegt innerhalb der Grenzen der Deskriptortabelle else
       #GP(Selektor+EXT)
     - Deskriptor (Accessbyte) beschreibt ein Kodesegment else
       #GP(Selektor+EXT)
     - Segment ist präsent else #NP(Selektor+EXT)
  - if Kodesegment "non-conforming" und DPL < CPL then

     Interrupt zu höherer Privilegierung

     - Test Selektor und Deskriptor für neuen Stack im aktuellen TSS:
        - Selektor ist ungleich null else #GP(EXT)
        - Selektor liegt innerhalb der Grenzen der Deskriptortabelle else
          #TS(SS-Selektor+EXT)
        - RPL des Selektors = DPL des Kodesegments else #TS(SS-Selek-
          tor+EXT)
        - DPL des Stacksegments = DPL des Kodesegments else #TS(SS-Selek-
          tor+EXT)
        - Deskriptor bezeichnet ein beschreibbares Datensegment else
          #TS(SS-Selektor+EXT)
        - Segment ist präsent else #SS(SS-Selektor+EXT)
     - im neuen Stack ist Platz für 10 Byte else #SS(0)
     - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
     - SS:SP := TSS
     - CS:IP := Gate
     - CS-Cache := Deskriptor
     - SS-Cache := Deskriptor
     - PUSH alten Stackpointer in den neuen Stack
     - PUSH Rückkehradresse in den neuen Stack
     - CPL := DPL des neuen Kodesegments
     - RPL-Feld des CS-Registers := CPL
     - if Interrupt-Gate:
        - IF := 0
     - TF := 0
     - NT := 0

  - if Kodesegment "conforming" oder DPL des Kodesegments = CPL then

     Interrupt in der gleichen Privilegierungsstufe

     - im aktuellen Stack ist Platz für 6 Bytes else #SS(0)
     - if Interrupt durch Exception mit Fehlerkode then
        - im aktuellen Stack ist Platz für 2 Bytes zusätzlich else #SS(0)
     - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
     - PUSH Flagregister
     - PUSH aktuellen CS-Selektor
     - PUSH Offset der Rückkehradresse
     - CS:IP := Gate
     - CS-Cache := Deskriptor
     - RPL-Feld des CS-Registers := CPL
```

```
      - PUSH Fehlerkode (falls vorhanden)
      - if Interrupt-Gate then
         - IF := 0
      - TF := 0
      - NT := 0

Else #GP(CS-Selektor+EXT)

- if Task-Gate then
   - Test TSS-Selektor im Task-Gatedeskriptor:
      - Selektor zeigt in GDT else #GP(TSS-Selektor)
      - Selektor liegt innerhalb der GDT-Grenzen else #GP(TSS-Selektor)
      - Deskriptor (Accessbyte) beschreibt ein verfügbares TSS else
        #GP(TSS-Selektor)
      - TSS ist präsent else #NP(TSS-Selektor)
   - Taskwechsel mit Verschachtelung zu TSS
   - if Interrupt durch Exception mit Fehlerkode then
      - im aktuellen Stack ist Platz für 2 Bytes zusätzlich else #SS(0)
      - PUSH Fehlerkode
   - IP liegt innerhalb des Kodesegmentlimits else #GP(0)

Anmerkung:
     Der Wert EXT im Fehlerkode ist eins, wenn ein externes Ereignis (z.B.
     ein Einzelschrittinterrupt, ein externer Interrupt, eine MF- oder MP-
     Exception) den Interrupt auslöste. Sonst hat EXT den Wert null.

Exceptions im RA-Mode: keine
     Der Mikroprozessor 80286 geht in den "shut-down"-Zustand, wenn der
     Wert im Stackpointerregister SP vor der Ausführung der Befehle INT
     oder INTO gleich 1, 3, oder 5 ist.
```

IRET Rückkehr aus einer Interruptbehandlungsroutine **IRET**

Opkode	Befehl	Takte	Beschreibung
CF	IRET	17,pm=31	Rückkehr vom Interrupt
CF	IRET	55	Rückkehr vom Interrupt, niedrigere Privilegstufe
CF	IRET	169	Rückkehr vom Interrupt, andere Task (NT=1)

Flags:
Der Inhalt des Flagregisters wird aus dem Stack rückgespeichert.

Operation:
Der Befehl IRET bewirkt die Fortsetzung des Programms, das durch eine externe Interruptanforderung, eine Exception oder einen INT-Befehl unterbrochen wurde, nachdem die Interruptbehandlungsroutine abgeschlossen ist. Er wird auch nach einem Taskwechsel mit Verschachtelung zur Rückkehr in die aufrufende Task benutzt (vgl. Befehl CALL über Task-Gate bzw. CALL über Task-State-Segment).
Im RA-Mode werden die bei der Interruptanerkennung in den Stack abgelegten Inhalte der Register IP, CS und des Flagregisters in der

genannten Reihenfolge zurückgespeichert.
Im PVA-Mode ist die Wirkung des IRET-Befehls vom Wert des Flags NT abhängig. Bei der Übernahme des Flagregisters aus dem Stack werden die Bitpositionen IOPL nur dann geändert, wenn die aktuelle Privilegierungsstufe CPL=0 ist.

NT=0: Rückkehr vom Interrupt ohne Taskwechsel.
Der Programmkode, in den die Rückkehr vom Interrupt erfolgt, muß die gleiche oder eine niedrigere Privilegierung als die Interruptbehandlungsroutine besitzen. Die Privilegierungsstufe des Programmkodes wird aus den Bitpositionen RPL des CS-Selektors entnommen, der aus dem Stack rückgespeichert wird. Ist diese Privilegierung niedriger - RPL des Selektors ist größer als CPL der Interruptroutine - werden durch den IRET-Befehl auch die Inhalte der Register SP und SS aus dem Stack geholt.

NT=1: Rückkehr vom Interrupt mit Taskwechsel.
Die durch den Interrupt aktivierte Task speichert ihren aktuellen Status im zugehörigen Task-State-Segment ab. Die Task, die ursprünglich vor dem Interrupt aktiv war, wird reaktiviert und weiter bearbeitet.

Exceptions im PVA-Mode:

- if NT=1 then

  Rückkehr aus verschachtelter Task

  - Test Back-Link-Selektor im aktuellen TSS:
    - TSS-Selektor zeigt in GDT else #TS(neuer TSS-Selektor)
    - TSS-Selektor liegt innerhalb der GDT-Grenzen else #TS(neuer TSS-Selektor)
    - Deskriptor (Accessbyte) beschreibt ein verfügbares TSS else #TS(neuer TSS-Selektor)
    - selektiertes TSS ist BUSY (Accessbyte) else #TS(neuer TSS-Selektor)
    - selektiertes TSS ist präsent else #NP(neuer TSS-Selektor)
  - Taskwechsel ohne Verschachtelung zum TSS (Back-Link-Selektor)
  - alter TSS-Deskriptor wird inaktiv gesetzt (not BUSY)
  - IP liegt innerhalb des Kodesegmentlimits else #GP(0)

- if NT=0 then

  Interruptrückkehr über Stack

  - 2 Worte oberhalb des Stackpointers liegen innerhalb des Stacksegmentlimits else #SS(0)
  - RPL des Rückkehrselektors >= CPL else #GP(Rückkehrselektor)
  - if RPL des Rückkehrselektors = CPL then

```
    Rückkehr zu gleicher Privilegierung

    - 6 Worte oberhalb des Stackpointers liegen innerhalb des  Stackseg-
      mentlimits else #SS(0)
    - Rückkehrselektor (SP+2) ist ungleich null else #GP(0)
    - Rückkehrselektor liegt innerhalb der Grenzen der Deskriptortabelle
      else #GP(Rückkehrselektor)
    - Deskriptor (Accessbyte) beschreibt ein Kodesegment else  #GP(Rück-
      kehrselektor)
    - if Kodesegment "non-conforming" then
       - DPL des Kodesegments = CPL else #GP(Rückkehrselektor)
    - if Kodesegment "conforming" then
       - DPL des Kodesegments <= CPL else #GP(Rückkehrselektor)
    - Programmkodesegment ist präsent else #NP(Rückkehrselektor)
    - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
    - CS:IP := Stack
    - CS-Cache := Deskriptor
    - Flagregister := Stack
    - SP := SP + 6

ELSE

    Rückkehr zu niedrigerer Privilegierung

    - 10 Worte oberhalb des Stackpointers liegen innerhalb des Stackseg-
      mentlimits else #SS(0)
    - Test Rückkehrselektor (SP+2) und zugehöriger Deskriptor:
       - Rückkehrselektor (SP+2) ist ungleich null else #GP(0)
       - Rückkehrselektor  liegt innerhalb der Grenzen  der  Deskriptor-
         tabelle else #GP(Rückkehrselektor)
       - Deskriptor  (Accessbyte)  beschreibt  ein  Kodesegment  else
         #GP(Rückkehrselektor)
       - if Kodesegment "non-conforming" then
          - DPL  des  Kodesegments  =  RPL  des  Rückkehrselektors  else
            #GP(Rückkehrselektor)
       - if Kodesegment "conforming" then
          - DPL des Kodesegments > CPL else #GP(Rückkehrselektor)
       - Programmkodesegment ist präsent else #NP(Rückkehrselektor)

    - Test SS-Selektor für Rückkehr (SP+8) und zugehöriger Deskriptor:
       - Selektor ist ungleich null else #GP(0)
       - Selektor liegt innerhalb der Grenzen der Deskriptortabelle else
         #GP(SS-Selektor)
       - RPL des Selektors = RPL des Kodesegments else #GP(SS-Selektor)
       - Deskriptor  (Accessbyte) bezeichnet ein  beschreibbares  Daten-
         segment else #GP(SS-Selektor)
       - DPL  des  Stacksegments  =  RPL  des  Rückkehrkodesegments  else
         #GP(SS-Selektor)
       - Stacksegment ist präsent else #NP(SS-Selektor)

    - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
    - CS:IP := Stack
    - Flagregister := Stack
```

```
      - SS:SP := Stack
      - CPL := RPL des Rückkehrkodesegmentselektors
      - CS-Cache := Deskriptor
      - SS-Cache := Deskriptor

      Für ES und DS gilt:
         - if Segmentregisterinhalt ist in der neuen Privilegierungsstufe
           nicht zulässig then
            - Segmentregister (Selektor und Accessbyte) := 0

         - Segmentregisterinhalt ist zulässig, wenn gilt:
            - Selektor liegt innerhalb der Grenzen der Deskriptortabelle
            - Deskriptor (Accessbyte) beschreibt ein Datensegment oder ein
              lesbares Kodesegment
            - if Datensegment oder Kodesegment "non-conforming" then
               - DPL >= CPL
               - DPL >= RPL

Exceptions im RA-Mode:
Interrupt 13   Offset des Stackpointers steht vor der Ausführung des IRET-
               Befehls auf OFFFFH
```

JMP Unbedingter Sprung **JMP**

Opkode	Befehl	Takte	Beschreibung
EB cb	JMP cb	7	Sprung SHORT (-128 bis +127 Byte)
E9 cw	JMP cw	7	Sprung NEAR (innerhalb des Segments)
FF /4	JMP ew	7,mem=11	Sprung NEAR indirekt (absolutes Offset)
EA cd	JMP cd	11,pm=23	Sprung FAR (4-Byte-Direktadresse)
EA cd	JMP cd	38	Sprung FAR über CALL-Gate, DPL=CPL
EA cd	JMP cd	175	Sprung FAR über TSS (Taskwechsel)
EA cd	JMP cd	180	Sprung FAR über Task-Gate (Taskwechsel)
FF /5	JMP ed	15,pm=26	Sprung FAR (4-Byte eff. Adr. im Speicher)
FF /5	JMP ed	41	Sprung FAR über CALL-Gate, DPL=CPL
FF /5	JMP ed	178	Sprung FAR über TSS (Taskwechsel)
FF /5	JMP ed	183	Sprung FAR über Task-Gate (Taskwechsel)

```
Flags:     OF DF IF TF SF ZF AF PF CF
           -  -  -  -  -  -  -  -  -

Ausnahme: Erfolgt ein Taskwechsel, werden die Flags der eintretenden Task
          aus dem Task-State-Segment geladen.
```

Operation:

Die Programmabarbeitung wird an einer anderen Programmadresse fortgesetzt, ohne eine Rückkehradresse (im Gegensatz zum Befehl CALL) abzuspeichern. Der Mikroprozessor 80286 kann in Abhängigkeit von der Betriebsart verschiedene JMP-Befehle ausführen:

- Sprünge innerhalb eines Segments:
  + JMP SHORT Distanzwert 8 Bit (-128 bis +127 Byte)
  + JMP NEAR Distanzwert 16 Bit (innerhalb eines 64-KByte-Segments)

- Sprünge in andere Segmente im RA-Mode:
  + JMP FAR : Angabe einer vollständigen Direktadresse

- Sprünge in andere Segmente im PVA-Mode:
  + JMP FAR vollständige Adresse in einem Programmkodesegment
  + JMP FAR vollständige Adresse eines CALL-Gates
  + JMP FAR vollständige Adresse eines Task-Gates
  + JMP FAR vollständige Adresse eines Task-State-Segments.

Alle Sprünge vom Typ NEAR und FAR können auch indirekt angegeben werden.

Der Befehl JMP SHORT wird für Sprünge im Nahbereich benutzt und spart Speicherplatz. Der im Befehl enthaltene 8-Bit-Direktwert wird intern vorzeichenbehaftet zu einem 16-Bit-Wert erweitert und zum Offset des dem Sprungbefehl folgenden Befehls modulo 65536 addiert.

Analog wird beim Befehl JMP NEAR ein vorzeichenbehafteter 16-Bit-Direktwert für die Offsetberechnung (Addition zum folgenden Befehl modulo 65536) verwendet. Wird beim Befehl JMP NEAR die indirekte Adressierung benutzt (Form JMP ew), ist die Offsetkomponente als absolutes Offset (bezogen auf den Anfang des 64-KByte-Segments) anzugeben.

Der Befehl JMP FAR im RA-Mode setzt die Register CS:IP auf den angegebenen Wert und führt dort die Programmabarbeitung fort.
Im PVA-Mode unterscheiden sich die Aktionen bei der Abarbeitung eines JMP FAR in Abhängigkeit vom Typ des Deskriptors, der durch die Selektorkomponente spezifiziert wird:

Selektor spezifiziert ein Programmkodesegment:
Ist das angegebene Segment präsent (im Speicher verfügbar) und befindet sich die Offsetkomponente innerhalb des Limits des Segments (adressierbar), werden CS:IP mit dem angegebene Sprungziel geladen.

Selektor spezifiziert ein CALL-Gate:
Nach verschiedenen Zugriffsüberprüfungen (siehe unten) werden CS:IP mit der Adresse geladen, die im CALL-Gate-Deskriptor angegeben ist. Die Offsetkomponente, die im Befehl enthalten sein muß, wird ignoriert.

Selektor spezifiziert ein Task-Gate:
Der Status der aktuellen Task wird im Task-State-Segment (TSS) gerettet. Anschließend wird das im Task-Gate adressierte TSS benutzt, um den Kontext der eintretenden Task zu laden. Die Verbindung beider Tasks wird über das Back-Link-Feld im TSS der eintretenden Task hergestellt. Das BUSY-Bit der bisher aktiven

Task im Accessbyte des TSS-Deskriptors wird rückgesetzt, das im TSS-Deskriptor der eintretenden Task wird gesetzt. Die Befehlsabarbeitung beginnt an dem Punkt, bei dem die eintretende Task zuletzt suspendiert wurde. Die Offsetkomponente, die im Befehl enthalten sein muß, wird ignoriert.

Selektor spezifiziert ein Task-State-Segment (TSS):
Der Taskwechsel erfolgt analog zu den eben erläuterten Aktionen. Der Unterschied liegt darin, daß der Selektor direkt auf den TSS-Deskriptor der eintretenden Task in der Globaldeskriptortabelle (GDT) zeigen muß.

Exceptions im PVA-Mode:
#GP(0) Die effektive Adresse befindet sich außerhalb des Segmentlimits des Programmkodesegments.
#UD Der Operand eines indirekten JMP FAR bezeichnet ein Register.

Aktionen bei Ausführung des Befehls JMP FAR:

- if Operand über indirekte Adressierung then
  - Zugriff auf effektive Doppelwortadresse ist erlaubt else #GP(0) oder #SS(0) bei Limitüberschreitung
- CS-Selektoroperand ist ungleich null else #GP(0)
- CS-Selektoroperand liegt innerhalb der Grenzen der Deskriptortabelle else #GP(Selektor)
- Test Accessbyte des durch den Selektor spezifizierten Deskriptors:

  JMP zu Programmkodesegment "conforming"

  - DPL <= CPL else #GP(CS-Selektor)
  - Segment ist präsent else #NP(CS-Selektor)
  - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
  - CS:IP := Pointeroperand
  - CS-Cache := Segmentdeskriptor

  JMP zu Programmkodesegment "non-conforming"

  - RPL <= CPL else #GP(CS-Selektor)
  - DPL = CPL else #GP(CS-Selektor)
  - Segment ist präsent else #NP(CS-Selektor)
  - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
  - CS:IP := Pointeroperand
  - CS-Cache := Segmentdeskriptor
  - RPL-Feld des CS-Registers := CPL

  JMP zu CALL-Gate

  - DPL des CALL-Gates >= CPL else #GP(CALL-Gateselektor)
  - DPL des CALL-Gates >= RPL des CALL-Gateselektors else #GP(CALL-Gateselektor)
  - CALL-Gate ist präsent else #NP(CALL-Gateselektor)
  - Test CS-Selektor im CALL-Gatedeskriptor:
    - Selektor ist ungleich null else #GP(0)

```
      - Selektor liegt innerhalb der Grenzen der Deskriptortabelle else
        #GP(CS-Selektor)
      - Deskriptor (Accessbyte) beschreibt ein Kodesegment else #GP(CS-
        Selektor)
      - if Programmkodesegment ist "non-conforming" then
         - DPL des Kodesegmentdeskriptors = CPL else #GP(CS-Selektor)
      - if Programmkodesegment ist "conforming" then
         - DPL des Kodesegmentdeskriptors <= CPL else #GP(CS-Selektor)
      - Programmkodesegment ist präsent else #NP(CS-Selektor)
   - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
   - CS:IP := CALL-Gate
   - CS-Cache := Segmentdeskriptor
   - RPL-Feld des CS-Registers := CPL

   JMP zu Task-Gate

   - DPL des Task-Gates >= CPL else #GP(Task-Gateselektor)
   - DPL des Task-Gates >= RPL des Task-Gateselektors else #GP(Task-
     Gateselektor)
   - Task-Gate ist präsent else #NP(Task-Gateselektor)
   - Test TSS-Selektor im Task-Gatedeskriptor:
      - TSS-Selektor zeigt in GDT else #GP(TSS-Selektor)
      - TSS-Selektor liegt innerhalb der GDT-Grenzen else #GP(TSS-
        Selektor)
      - Deskriptor (Accessbyte) beschreibt ein verfügbares TSS else
        #GP(TSS-Selektor)
      - TSS ist präsent else #NP(TSS-Selektor)
   - Taskwechsel ohne Verschachtelung zu TSS
   - IP liegt innerhalb des Kodesegmentlimits else #GP(0)

   JMP zu Task-State-Segment

   - DPL des TSS >= CPL else #GP(TSS-Selektor)
   - DPL des TSS >= RPL des TSS-Selektors else #GP(TSS-Selektor)
   - Deskriptor (Accessbyte) beschreibt ein verfügbares TSS else #GP(TSS-
     Selektor)
   - TSS ist präsent else #NP(TSS-Selektor)
   - Taskwechsel ohne Verschachtelung zu TSS
   - IP liegt innerhalb des Kodesegmentlimits else #GP(0)

ELSE #GP(CS-Selektor)

Exceptions im RA-Mode:
#UD      Der Operand eines indirekten JMP FAR bezeichnet ein Register.
```

Jbed Bedingte relative Sprünge **Jbed**

Opkode	Befehl	Takte	Beschreibung
70 cb	JO cb	7+a,noj=3	rel. Sprung (overflow, OF=1)
71 cb	JNO cb	7+a,noj=3	rel. Sprung (not overflow, OF=0)
72 cb	JC cb	7+a,noj=3	rel. Sprung (carry, CF=1)
72 cb	JB cb	7+a,noj=3	rel. Sprung (below, CF=1)
72 cb	JNAE cb	7+a,noj=3	rel. Sprung (not above/equal, CF=1)
73 cb	JNC cb	7+a,noj=3	rel. Sprung (no carry, CF=0)
73 cb	JNB cb	7+a,noj=3	rel. Sprung (not below, CF=0)
73 cb	JAE cb	7+a,noj=3	rel. Sprung (above equal, CF=0)
74 cb	JZ cb	7+a,noj=3	rel. Sprung (zero, ZF=1)
74 cb	JE cb	7+a,noj=3	rel. Sprung (equal, ZF=1)
75 cb	JNZ cb	7+a,noj=3	rel. Sprung (not zero, ZF=0)
75 cb	JNE cb	7+a,noj=3	rel. Sprung (not equal, ZF=0)
76 cb	JNA cb	7+a,noj=3	rel. Sprung (not above, CF=1 oder ZF=1)
76 cb	JBE cb	7+a,noj=3	rel. Sprung (below equal, CF=1 oder ZF=1)
77 cb	JA cb	7+a,noj=3	rel. Sprung (above, CF=0 und ZF=0)
77 cb	JNBE cb	7+a,noj=3	rel. Sprung (not below/equal, CF=0 und ZF=0)
78 cb	JS cb	7+a,noj=3	rel. Sprung (sign, SF=1)
79 cb	JNS cb	7+a,noj=3	rel. Sprung (not sign, SF=0)
7A cb	JP cb	7+a,noj=3	rel. Sprung (parity, PF=1)
7A cb	JPE cb	7+a,noj=3	rel. Sprung (parity even, PF=1)
7B cb	JNP cb	7+a,noj=3	rel. Sprung (not parity, PF=0)
7B cb	JPO cb	7+a,noj=3	rel. Sprung (parity odd, PF=0)
7C cb	JL cb	7+a,noj=3	rel. Sprung (less, SF!=OF)
7C cb	JNGE cb	7+a,noj=3	rel. Sprung (not greater/equal, SF!=OF)
7D cb	JNL cb	7+a,noj=3	rel. Sprung (not less, SF=OF)
7D cb	JGE cb	7+a,noj=3	rel. Sprung (greater equal, SF=OF)
7E cb	JLE cb	7+a,noj=3	rel. Sprung (less equal, ZF=1 oder SF!=OF)
7E cb	JNG cb	7+a,noj=3	rel. Sprung (not greater, ZF=1 oder SF!=OF)
7F cb	JNLE cb	7+a,noj=3	rel. Sprung (not less/equal, ZF=0 und SF=OF)
7F cb	JG cb	7+a,noj=3	rel. Sprung (greater, ZF=0 und SF=OF)
E3 cb	JCXZ cb	8+a,noj=4	rel. Sprung, wenn CX=0

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Bedingte Sprünge werden für Programmverzweigungen in Abhängigkeit von den Werten des Flagregisters benutzt, das durch vorangegangene Operationen beeinflußt wurde. Die Bedingung für die Ausführung des Sprungs ist in der folgenden Tabelle in Klammern angegeben, sie ist jedoch auch aus der Abkürzung im mnemonischen Operationskode erkennbar.

Einige Buchstabenkombinationen stehen für die gleiche Sprungbedingung. Hier kann der Programmierer aus programmlogischen Gesichtspunkten die passenden Befehle auswählen. Zum Beispiel wird der Befehl JE günstigerweise dann benutzt, wenn nach Berechnungen die Gleichheit zweier Werte überprüft werden soll, während der Befehl JZ bei logischen Operationen exakter ist. Beide Befehle führen jedoch die gleiche Funktion aus.

```
          A   vorzeichenlos größer (above)
          B   vorzeichenlos kleiner (below)
          C   Übertrag (carry)
          E   gleich, gerade (equal, even)
          G   vorzeichenbehaftet größer (greater)
          L   vorzeichenbehaftet kleiner (less)
          N   nicht (not)
          O   Überlauf, ungerade (overflow, odd)
          P   Parität (parity)
          S   Vorzeichen (sign)
          Z   Null (zero)
```

Ist die angegebene Bedingung erfüllt, wird die Verzweigung zu dem Befehl auf der Adresse vollzogen, die sich durch Addition des aktuellen Inhalts des IP-Registers (d.h., IP zeigt während der Ausführung auf den nach dem Sprungbefehl folgenden Befehl) und des im bedingten Sprungbefehl angegebenen vorzeichenbehafteten Distanzbytes ergibt.

Der Befehl JCXZ testet den Inhalt des Registers CX (keine Flagbedingung). Enthält CX den Wert 0, wird der Sprung ausgeführt. Dieser Befehl kann günstig zur Schleifensteuerung bei Zeichenkettenverarbeitungen verwendet werden.

Da der Distanzwert beim Mikroprozessor 80286 nur in Bytegröße angebbar ist, können bedingte Sprünge nur im Bereich von -128 bis +127 Byte bezogen auf den Inhalt des IP-Registers ausgeführt werden. Liegt das Sprungziel außerhalb dieses Bereichs oder befindet es sich in einem anderen Programmkodesegment (FAR Label), muß ein bedingter Sprung mit der entgegengesetzten Sprungbedingung über einen unbedingten Sprung, der die Verzweigung realisiert, verwendet werden ("jump-around-jump"-Methode).

Beispiel für die "jump-around-jump"-Methode

Nach einem Vergleich wird zur Marke m_far in einem anderen Programmkodesegment verzweigt, wenn der Wert in AX größer als 20 ist.

```
          CMP  AX, 20     ;Vergleich
          JLE  m1         ;relativer Sprung mit entgegengesetzter
                          ;Sprungbedingung über den gewünschten Sprung-
                          ;befehl hinweg
          JMP  m_far      ;Sprung in das andere Programmkodesegment
     m1:
                          ;nächster Befehl
```

Exceptions im PVA-Mode:
#GP(0) Die resultierende Offsetkomponente des Sprungziels liegt außerhalb des Programmkodesegments

Exceptions im RA-Mode: keine

LAHF Laden der Flags nach AH **LAHF**

Opkode	Befehl	Takte	Beschreibung
9F	LAHF	2	AH SF ZF xx AF xx PF xx CF

Flags:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
-	-	-	-	-	-	-	-	-

Operation:

Der Inhalt des Low-Bytes des Flagregisters wird in der oben gezeigten Form in das Register AH übertragen. Die durch xx gekennzeichneten Bitpositionen enthalten undefinierte Werte (vgl. Befehl SAHF).

Keine Exceptions.

LAR Laden des Accessbytes **LAR**

Opkode	Befehl	Takte	Beschreibung
0F 02 /r	LAR rw,ew	14,mem=16	high(rw) := Accessbyte(Selektor ew)

Flags:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
-	-	-	-	-	M	-	-	-

Operation:

LAR lädt das Accessbyte aus dem Deskriptor des Selektors, der durch den zweiten Operanden angegeben ist, in den High-Teil des durch den ersten Operanden angegebenen Wortregisters. Das Low-Byte des Wortregisters wird zu null gesetzt.
Voraussetzung ist, daß sowohl die aktuelle Privilegierungsstufe CPL als auch die durch RPL des Deskriptors festgelegte Privilegierungsstufe den Zugriff auf den Deskriptor erlauben (DPL >= CPL und DPL >= RPL) und der Selektor innerhalb der Grenzen der Deskriptortabelle liegt.
Konnte das Accessbyte geladen werden, ist das Flag ZF=1 gesetzt, sonst gilt ZF=0.

Es ist zu beachten, daß durch den Zugriff über den Selektoroperanden keine Exception resultieren kann. Exceptions werden nur generiert, wenn der Speicheroperand, der den Selektor enthält, nicht zugreifbar ist.

Exceptions im PVA-Mode:

#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 6	Der Befehl LAR ist im RA-Mode nicht zulässig.

LDS/LES Laden eines Doppelwortpointers **LDS/LES**

Opkode	Befehl	Takte	Beschreibung
C5 /r	LDS rw,ed	7,pm=21	DS:rw := ed
C4 /r	LES rw,ed	7,pm=21	ES:rw := ed

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Die Befehle LDS bzw. LES laden ein Segmentregister und ein Wortregister mit dem Inhalt eines im Speicher befindlichen 4-Byte-Pointeroperanden, der durch den zweiten Operanden des Befehls spezifiziert wird. Das erste Wort des Pointers (Offsetkomponente auf Speicheradresse ed) wird in das Wortregister geladen, das zweite Wort (Selektorkomponente auf Speicheradresse ed+2) wird in das Segmentregister übertragen, das im mnemonischen Operationskode bezeichnet ist.

Im PVA-Mode wird der unsichtbare Teil des Segmentregisters aus dem durch den Selektor spezifizierten Deskriptoreintrag aus der GDT bzw. LDT ebenfalls geladen.

Es ist möglich, einen Nullselektor in eines der Segmentregister zu laden, ohne daß eine Exception generiert wird. Versucht jedoch ein Programm über diesen Selektor (d.h. das Segmentregister mit dem Inhalt null) auf den Speicher zuzugreifen, wird eine General-Protection-Exception (#GP(0)) ausgelöst, ohne daß der Speicherzugriff erfolgte.

Exceptions im PVA-Mode:

- if Selektor ist ungleich null then
  - neuer Selektor liegt innerhalb der Grenzen der Deskriptortabelle else #GP(Selektor)
  - Test Accessbyte des Deskriptors:
    - if Datensegment oder lesbares Programmkodesegment "non-conforming" then
      - DPL >= CPL else #GP(Selektor)
      - DPL >= RPL des Selektors else #GP(Selektor)
    - if lesbares Programmkodesegment "conforming" then
      - keine Überprüfungen von DPL, RPL oder CPL

    else #GP(Selektor)

  - Segment ist präsent else #NP(CS-Selektor)
  - Register := Offset aus Pointeroperand (erstes Wort)
  - Segmentregister := Selektor aus Pointeroperand (zweites Wort)
  - Segmentregister-Cache := Segmentdeskriptor

- if Selektor gleich null then
  - Register := Offset aus Pointeroperand (erstes Wort)
  - Segmentregister := Selektor aus Pointeroperand (zweites Wort)

- Segmentregister-Cache wird als ungültig markiert

#GP(0) und #SS(0)	Der Operand liegt außerhalb des Segmentlimits.
#UD	Der Quelloperand ist ein Register.

Exceptions im RA-Mode:

Interrupt 13	Die Offsetkomponente der effektiven Adresse hat den Wert OFFFDH oder OFFFFH.
#UD	Der Quelloperand ist ein Register.

LEA — Laden des effektiven Adreßoffsets — LEA

Opkode	Befehl	Takte	Beschreibung
8D /r	LEA rw,m	3	rw := OFFSET(m)

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Die effektive Adresse (Offsetkomponente) des zweiten Operanden (allgemeiner Speicheroperand) wird im Wortregister übergeben, das durch den ersten Operanden spezifiziert wurde.

Exceptions im PVA-Mode wie RA-Mode:

#UD	Der zweite Operand bezeichnet ein Register.

LEAVE — Verlassen eine Stackrahmens bei Prozeduraustrit — LEAVE

Opkode	Befehl	Takte	Beschreibung
C9	LEAVE	5	SP := BP ; Verlassen lokaler Variablen BP := (SP) ; POP BP SP := SP + 2

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl LEAVE verläßt die durch den Befehl ENTER angelegte Stackrahmenstruktur und stellt den Zustand der Register BP und SP, der vor der Ausführung des letzten ENTER-Befehls bestand, wieder her.

Ein nachfolgender Befehl RET n gibt anschließend den durch eventuelle Parameterübergabe belegten Stackbereich oberhalb des Stackrahmens und der Rückkehradresse wieder frei (vgl. Befehl ENTER).

Exceptions im PVA-Mode:
#SS(0) BP zeigt nicht auf eine zulässige Adresse im aktuellen Stacksegment.

Exceptions im RA-Mode:
Interrupt 13 Der Wortoperand besitzt ein Offset von 0FFFFH.

LGDT/LIDT Laden der Register GDTR/IDTR **LGDT/LIDT**

Opkode	Befehl	Takte	Beschreibung
0F 01 /2	LGDT m	11.	GDTR := m m+5
0F 01 /3	LIDT m	12	IDTR := m m+5

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:
Das Globaldeskriptortabellenregister (GDTR) bzw. das Interruptdeskriptortabellenregister (IDTR) werden mit den Werten (6 Bytes) geladen, die beginnend mit der angegebenen effektiven Adresse im Speicher bereitgestellt wurden. Der sechs Byte große Speicherblock muß folgenden Inhalt haben:

m+0: Länge der Deskriptortabelle (Bit 7-0)
m+1: Länge der Deskriptortabelle (Bit 15-8)
m+2: Basisadresse der Deskriptortabelle (Bit 7-0)
m+3: Basisadresse der Deskriptortabelle (Bit 15-8)
m+4: Basisadresse der Deskriptortabelle (Bit 23-16)
m+5: Null

Das letzte Byte wird vom Mikroprozessor 80286 ignoriert.
Die Befehle LGDT und LIDT sind nur in Betriebssystemprogrammen üblich. Sie können sowohl im RA-Mode als auch im PVA-Mode zur Initialisierung der Speicherverwaltung bzw. des Interruptsystems für den PVA-Mode verwendet werden. Im PVA-Mode sind dies die einzigen Befehle des Mikroprozessors 80286, die physische Adressen laden (vgl. Befehle SGDT und SIDT).

Exceptions im PVA-Mode:
#GP(0) Die aktuelle Privilegierungsstufe ist nicht null.
#UD Der Quelloperand ist ein Register.
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13 Der Wortoperand besitzt ein Offset von 0FFFFH.
#UD Der Quelloperand ist ein Register.

LLDT Laden des Registers LDTR **LLDT**

Opkode	Befehl	Takte	Beschreibung
0F 00 /2	LLDT ew	17,mem=19	LDTR := Deskriptor(ew)

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl LLDT dient zum Laden des Lokaldeskriptortablellenregisters LDTR. Der Wortoperand muß einen Selektor enthalten, der auf einen LDT-Deskriptor in der Globaldeskriptortabelle (GDT) zeigt. Ist dies der Fall, wird der sichtbare Teil des Registers LDTR mit dem Selektor und der Cache-Teil des LDTR mit den Werten des LDT-Deskriptors aus der GDT geladen.

Enthält der Selektoroperand den Wert 0, wird das LDTR als ungültig markiert. In diesem Fall führen alle Referenzen über das LDTR zu einer General-Protection-Exception (Ausnahmen bilden die Befehle LAR, VERR, VERW und LSL).

Der Befehl LLDT ist für Betriebssystemprogramme vorgesehen (vgl. Befehl SLDT).

Exceptions im PVA-Mode:

#GP(0)	Die aktuelle Privilegierungsstufe ist nicht null.
#GP(selektor)	Der Selektoroperand zeigt nicht in die GDT.
#GP(selektor)	Der Eintrag in der GDT ist kein LDT-Deskriptor.
#NP(selektor)	Der LDT-Deskriptor ist nicht präsent.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 6	Der Befehl LLDT ist im RA-Mode nicht zulässig.

LMSW Laden des Maschinenstatusworts **LMSW**

Opkode	Befehl	Takte	Beschreibung
0F 01 /6	LMSW ew	3,mem=6	MSW := ew

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Quelloperand wird in das Maschinenstatuswort geladen (vgl. Befehl SMSW). Mit diesem privilegierten Befehl kann z.B. der Übergang vom RA-Mode in den PVA-Mode vollzogen werden. Ein Übergang vom PVA-Mode in den RA-Mode ist mit diesem Befehl nicht möglich (nur ein Hardware-RESET realisiert diesen Übergang, siehe auch Abschnitt 3.).

Achtung!

Sofort nach dem Übergang in den PVA-Mode muß ein Sprungbefehl JMP NEAR oder JMP SHORT ausgeführt werden, um die interne Befehlswarteschlange zu leeren! Nur so ist sichergestellt, daß nach der Umschaltung alle Befehle in der neu eingestellten Betriebsart richtig interpretiert und ausgeführt werden.

Der Befehl dient auch zur Spezifikation der numerischen Verarbeitungsvariante (Numerikcoprozessor oder dessen Emulation) in einem Softwaresystem (Abschn. 4.5. Maschinenstatuswort).

Exceptions im PVA-Mode:
#GP(0) Die aktuelle Privilegierungsstufe ist nicht null.
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13 Der Wortoperand besitzt ein Offset von OFFFFH.

LOCK — Aktivieren des Bussignals /LOCK — LOCK

Opkode	Befehl	Takte	Beschreibung
F0	LOCK	0	Aktivieren des Bussignals /LOCK

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

LOCK ist ein Präfix, der die Aktivierung des Bussignals /LOCK während der Ausführung des folgenden Befehls bewirkt. Es wird damit gesichert, daß in einem Multiprozessorsystem ein für mehrere Prozessoren zugänglicher Speicherbereich für die Zeitdauer der Befehlsausführung ausschließlich nur für diesen einen Prozessor zur Verfügung steht.

Der Präfix LOCK kann für die Befehle MOVS, INS und OUTS angewendet werden. Für die Ausführung des Befehls XCHG wird das Bussignal /LOCK immer aktiviert, unabhängig davon, ob der LOCK-Präfix angegeben wurde oder nicht.

Exceptions im PVA-Mode:
#GP(0) Die aktuelle Privilegierungsstufe ist höher (niedrigere Priorität) als die Stufe, die durch die Bitpositionen IOPL im Flagregister angegeben ist.

Exceptions im RA-Mode: keine

LODS Laden von Zeichenkettenoperanden **LODS**

Opkode	Befehl	Takte	Beschreibung	
AC	LODS mb	5	AL := [SI]	; SI setzen
AD	LODS mw	5	AX := [SI]	; SI setzen
AC	LODSB	5	AL := DS:[SI]	; SI setzen
AD	LODSW	5	AX := DS:[SI]	; SI setzen

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl LODS lädt das Element einer Zeichenkette, das durch das Register SI adressiert wird, in den Akkumulator (AL bzw. AX). Nach dem Transfer wird das Register SI auf das nächste Element der Zeichenkette gesetzt. Durch das Flag DF wird dabei bestimmt, ob das nächste Element der Zeichenkette in steigender oder fallender Adreßrichtung spezifiziert wird. Die Elementgröße (Byte oder Wort) wird entweder durch den mnemonischen Operationskode oder durch den Operandentyp angegeben und beeinflußt die Veränderung des Zeichenpointers SI (siehe Tabelle).

	DF=0	DF=1
LODSB	SI := SI + 1	SI := SI - 1
LODSW	SI := SI + 2	SI := SI - 2

In den ersten beiden Formen des Befehls kann ein Segment-Override-Präfix angegeben werden, während in den letzten beiden Formen (LODSB, LODSW) das Segmentregister DS voreingestellt ist.

Exceptions im PVA-Mode:

#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

LOOP Steuerung von Programmschleifen **LOOP**

Opkode	Befehl	Takte	Beschreibung
E2 cb	LOOP cb	8,noj=4	CX := CX-1; Sprung, wenn CX!=0
E1 cb	LOOPE cb	8,noj=4	CX := CX-1; Sprung, wenn CX!=0 und ZF=1
E1 cb	LOOPZ cb	8,noj=4	identisch LOOPE
E0 cb	LOOPNE cb	8,noj=4	CX := CX-1; Sprung, wenn CX!=0 und ZF=0
E0 cb	LOOPNZ cb	8,noj=4	identisch LOOPNE

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Die verschiedenen LOOP-Befehle werden zur Steuerung von Programmschleifen benutzt. Das Register CX dient dabei als Schleifenzähler und sollte vor Eintritt in die Schleife mit der Anzahl der Iterationen geladen werden.
Der Befehl LOOP dekrementiert zuerst das Register CX, ohne Flags zu beeinflussen. Ist anschließend das Register CX ungleich null, wird ein Sprung innerhalb des aktuellen Programmkodesegments ausgeführt. Das Sprungziel, das auf den Eintrittspunkt der Schleife gerichtet sein sollte, kann im Bereich von -128 bis +127 Byte, gerechnet ab Adresse des folgenden Befehls, liegen. Der Distanzwert, der im Befehl angegeben ist, wird vor der Addition intern vorzeichenbehaftet zu einem 16-Bit-Wert erweitert. Ist die Sprungbedingung nicht erfüllt, wird der dem LOOP-Befehl folgende Befehl ausgeführt.

Die Befehle LOOPE (entspricht LOOPZ) und LOOPNE (entspricht LOOPNZ) werten neben dem Inhalt des Registers CX das Flag ZF zur Ermittlung der Sprungbedingung aus (s. obige Übersicht).

Exceptions im PVA-Mode:
#GP(0) Die Offsetkomponente des Sprungziels liegt außerhalb des Limits des aktuellen Programmkodesegments.

Exceptions im RA-Mode: keine

LSL Laden des Segmentlimits **LSL**

Opkode	Befehl	Takte	Beschreibung
0F 03 /r	LSL rw,ew	14,mem=16	rw := Segmentlimit(Selektor ew)

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	M	-	-	-

Operation:

Der Befehl LSL übergibt die Information über das Limit aus dem Deskriptoreintrag eines durch den Selektor (zweiter Operand) spezifizierten Segments zur Auswertung in einem Register. Zu einer erfolg-

reichen Abarbeitung müssen folgende Voraussetzung erfüllt sein:

- Der Selektor muß ungleich null sein.
- Der Selektor muß einen Deskriptor innerhalb des Limits der Deskriptortabelle spezifizieren.
- DPL (Deskriptor) >= CPL ;(Ausnahme: conforming Kodesegment).
- DPL (Deskriptor) >= RPL (Selektor).

Sind all diese Voraussetzungen erfüllt, enthält das Register nach der Befehlsausführung den Inhalt des Limitfeldes des selektierten Deskriptors, und das Flag ZF ist gesetzt. Ist eine Voraussetzung nicht erfüllt, gilt ZF=0, und der Inhalt des Registers ist undefiniert.
Der Befehl LSL gibt nur das Limitfeld von Speichersegmenten, Task-Status-Segmenten und Lokaldeskriptortabellen zurück. Die Interpretation des Limitfeldes ist vom Typ des Segments abhängig.

Exceptions im PVA-Mode:

#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 6	Der Befehl LSL ist im RA-Mode nicht zulässig.

LTR — Laden des Taskregisters — LTR

Opkode	Befehl	Takte	Beschreibung
0F 00 /3	LTR ew	17,mem=19	TR := ew

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Mit dem Befehl LTR wird das Taskregister des 80286 mit einem Selektor geladen, der auf einen TSS-Deskriptor in der Globaldeskriptortabelle (GDT) zeigen muß (Abschn. 8.3. Taskregister). Das selektierte Task-State-Segment wird durch Setzen des BUSY-Bits im Accessbyte des TSS-Deskriptors als "besetzt" gekennzeichnet.

Achtung!

Der privilegierte Befehl LTR löst keinen Taskwechsel aus. Er dient hauptsächlich zum erstmaligen Laden des Taskregisters für die Starttask eines Betriebssystems. Nach dem Aktivieren der Starttask im PVA-Mode setzt die 80286-CPU automatisch bei jedem Taskwechsel das Taskregister auf das Task-State-Segment der eintretenden Task.
Der Inhalt des sichtbaren Teils des Taskregisters kann über den Befehl STR für weitere Auswertungen bereitgestellt werden.

```
Exceptions im PVA-Mode:
#GP(0)          Die aktuelle Privilegierungsstufe (CPL) ist nicht null.
#GP(0)          Die effektive Adresse eines Speicheroperanden in einem der
                Segmente CS, DS oder ES ist nicht zulässig.
#GP(sel)        Der Selektor zeigt nicht auf ein TSS, oder das TSS ist
                bereits besetzt (busy).
#SS(0)          Die Adresse im Segment SS ist nicht zulässig.
#NP             Das TSS ist nicht präsent.

Exceptions im RA-Mode:
Interrupt 6     Der Befehl LTR ist im RA-Mode nicht zulässig.
```

MOV Ladebefehl **MOV**

Opkode	Befehl	Takte	Beschreibung
88 /r	MOV eb,rb	2,mem=3	eb := rb
89 /r	MOV ew,rw	2,mem=3	ew := rw
8A /r	MOV rb,eb	2,mem=5	rb := eb
8B /r	MOV rw,ew	2,mem=5	rw := ew
8C /0	MOV ew,ES	2,mem=3	ew := ES
8C /1	MOV ew,CS	2,mem=3	ew := CS
8C /2	MOV ew,SS	2,mem=3	ew := SS
8C /3	MOV ew,DS	2,mem=3	ew := DS
8E /0	MOV ES,mw	5,pm=19	ES := mw
8E /0	MOV ES,rw	2,pm=17	ES := rw
8E /2	MOV SS,mw	5,pm=19	SS := mw
8E /2	MOV SS,rw	2,pm=17	SS := rw
8E /3	MOV DS,mw	5,pm=19	DS := mw
8E /3	MOV DS,rw	2,pm=17	DS := rw
A0 dw	MOV AL,xb	5	AL := xb (ohne Basis- oder Indexadr.)
A1 dw	MOV AX,xw	5	AX := xw (ohne Basis- oder Indexadr.)
A2 dw	MOV xb,AL	3	xb := AL (ohne Basis- oder Indexadr.)
A3 dw	MOV xb,AX	3	xw := AX (ohne Basis- oder Indexadr.)
B0+ rb db	MOV rb,db	2	rb := db
B8+ rw dw	MOV rw,dw	2	rw := dw
C6 /0 db	MOV eb,db	2,mem=3	eb := db
C7 /0 dw	MOV ew,dw	2,mem=3	ew := dw

```
Flags:    OF DF IF TF SF ZF AF PF CF
          -  -  -  -  -  -  -  -  -
```

Operation:

Der Befehl MOV kopiert ein Byte bzw. ein Wort von einer Quelladresse (zweiter Operand) zu einer Zieladresse (erster Operand). Ist als Zieloperand ein Segmentregister (DS, ES oder SS) angegeben, wird auch der nicht sichtbare Teil des Segmentregisters (Cache) mit dem Inhalt des durch den Quelloperand (Selektor) spezifizierten Segmentdeskriptors aus der Globaldeskriptortabelle (GDT) bzw. Lokaldeskriptortabelle (LDT) geladen.

Es ist möglich, einen Nullselektor in eines der Segmentregister DS oder ES zu laden, ohne daß es zu einer Verletzung der Zugriffsrechte kommt. Hingegen führt jede Referenz über ein Segmentregister mit einem Nullselektor zu einer General-Protection-Exception (#GP(0)), ohne daß auf den Speicher zugegriffen wird.

Wird das Segmentregister SS geladen, sind alle Interrupts gesperrt, bis der nächstfolgende Befehl vollständig ausgeführt wurde.

Exceptions im PVA-Mode:

- if SS := Selektor then
  - Selektor ist ungleich null else #GP(0)
  - Selektor liegt innerhalb der Grenzen der Deskriptortabelle else #GP(Selektor)
  - RPL des Selektors = CPL else #GP(Selektor)
  - Deskriptor (Accessbyte) bezeichnet ein beschreibbares Datensegment else #GP(Selektor)
  - DPL des Deskriptors = CPL else #GP(Selektor)
  - Segment ist präsent else #SS(SS-Selektor)
  - SS := Selektor
  - SS-Cache := Segmentdeskriptor

- if DS oder ES := Selektor ungleich null then
  - Selektor liegt innerhalb der Grenzen der Deskriptortabelle else #GP(Selektor)
  - Deskriptor (Accessbyte) beschreibt ein Datensegment oder lesbares Kodesegment else #GP(Selektor)
  - if Datensegment oder Kodesegment "non-conforming" then
    - DPL >= CPL else #GP(Selektor)
    - DPL >= RPL des Selektors else #GP(Selektor)
  - Segment ist präsent else #NP(Selektor)
  - Segmentregister := Selektor
  - Segmentregister-Cache := Segmentdeskriptor

- if DS oder ES := Nullselektor then
  - Segmentregister := Selektor
  - Segmentregister-Cache wird als ungültig markiert (Accessbyte)

#GP(0)	Der Zieloperand befindet sich in einem schreibgeschützten Segment
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

MOVS — Transfer von Zeichenketten — MOVS

Opkode	Befehl	Takte	Beschreibung
A4	MOVS mb,mb	5	ES:[DI] := [SI] ; SI, DI setzen
A5	MOVS mw,mw	5	ES:[DI] := [SI] ; SI, DI setzen
A4	MOVSB	5	ES:[DI] := DS:[SI] ; SI, DI setzen
A5	MOVSW	5	ES:[DI] := DS:[SI] ; SI, DI setzen

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl MOVS dient zum Kopieren von Zeichenketten. Der Inhalt der durch das Register SI adressierten Speicherzelle wird in die durch den Pointer ES:DI adressierten Speicherzelle geladen. Der Zieloperand muß über das Segmentregister ES adressiert werden. Die Angabe eines Segment-Override-Präfix ist nur für den Quelloperanden möglich.
Nach den Datentransfer werden die Register SI und DI automatisch so gesetzt, daß sie jeweils auf das nächste Element der Zeichenkette zeigen. Die Richtung, in der die Zeichenkette behandelt wird, ist durch das Flag DF (direction flag) vorgegeben.

	DF=0	DF=1
MOVSB	SI:=SI+1 ; DI:=DI+1	SI:=SI-1 ; DI:=DI-1
MOVSW	SI:=SI+2 ; DI:=DI+2	SI:=SI-2 ; DI:=DI-2

Mit Hilfe des Präfixes REP kann mit einem Befehl eine Zeichenkette der Länge, die durch das Register CX vorgegeben ist (maximal 64 KByte), umgeladen werden.

Exceptions im PVA-Mode:

#GP(0)	Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

MUL — Vorzeichenlose Multiplikation — MUL

Opkode	Befehl	Takte	Beschreibung
F6 /4	MUL eb	13,mem=16	AX = AL * eb
F7 /4	MUL ew	21,mem=24	DX:AX = AX * ew

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	M	-	-	-	U	U	U	U	M

Operation:

Der Quelloperand wird mit dem Akkumulator vorzeichenlos multipliziert. Das Ergebnis wird im Akkumulator AX (bei Byteoperanden) bzw. in den Registern DX:AX (bei Wortoperanden; DX enthält den höherwertigen Teil) abgelegt. Enthält der höherwertige Teil des Ergebnisses in jeder Bitposition den Wert 0, werden die Flags CF und OF auf Null gesetzt. Anderenfalls haben sie den Wert 1.

Quell-operand	Operation	CF / OF = 0 , wenn gilt:
Byte	AX = AL * eb	AH = 0 oder AH = 0FFH
Wort	DX:AX = AX * ew	DX = 0 oder DX = 0FFFFH

Sollen vorzeichenlose Zahlen mit Direktoperanden multipliziert werden, ist der Befehl IMUL anwendbar.

Exceptions im PVA-Mode:

#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von 0FFFFH.

NEG — Negation (Zweierkomplementbildung) — NEG

Opkode	Befehl	Takte	Beschreibung
F6 /3	NEG eb	2,mem=7	eb := 0 - eb
F7 /3	NEG ew	2,mem=7	ew := 0 - ew

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	M	-	-	-	M	M	M	M	M

Operation:

Der Befehl NEG bildet das Zweierkomplement des Operanden und überschreibt mit dem Ergebnis den Quelloperanden. Das Flag CF ist immer gesetzt, wenn der Quelloperand ungleich null ist, anderenfalls ist es rückgesetzt.

Exceptions im PVA-Mode:

#GP(0)	Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von 0FFFFH.

NOP Keine Operation **NOP**

Opkode	Befehl	Takte	Beschreibung
90	NOP	3	keine Operation

Flags: OF DF IF TF SF ZF AF PF CF
- - - - - - - - -

Operation:
Der 1-Byte-Befehl NOP führt keine Operation aus. Er wird in Zeitschleifen oder zur Speicherplatzreservierung für eventuelle spätere Manipulationen benutzt.

Keine Exceptions.

NOT Logische Invertierung (Einerkomplementbildung) **NOT**

Opkode	Befehl	Takte	Beschreibung
F6 /2	NOT eb	2,mem=7	eb := /eb
F7 /2	NOT ew	2,mem=7	ew := /ew

Flags: OF DF IF TF SF ZF AF PF CF
- - - - - - - - -

Operation:
Der Befehl NOT realisiert eine bitweise logische Invertierung des Quelloperanden (Einerkomplement, logische Negation). Das Ergebnis überschreibt den Quelloperanden.

Exceptions im PVA-Mode:

#GP(0)	Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

OR Logisches ODER **OR**

Opkode	Befehl	Takte	Beschreibung
08 /r	OR eb,rb	2,mem=7	eb := eb ODER rb
09 /r	OR ew,rw	2,mem=7	ew := ew ODER rw
0A /r	OR rb,eb	2,mem=7	rb := rb ODER eb
0B /r	OR rw,ew	2,mem=7	rw := rw ODER ew
0C db	OR AL,db	3	AL := AL ODER db
0D dw	OR AX,dw	3	AX := AX ODER dw
80 /1 db	OR eb,db	3,mem=7	eb := eb ODER db
81 /1 dw	OR ew,dw	3,mem=7	ew := ew ODER dw

Flag:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	0	-	-	-	M	M	U	M	0

Operation:

Der Befehl realisiert eine bitweise logische ODER-Verknüpfung der Operanden. Enthalten korrespondierende Bitpositionen beider Operanden den Wert 0, wird das Resultatbit ebenfalls auf den Wert 0 gesetzt. In jedem anderen Fall erhält das Resultatbit den Wert 1. Das Resultat wird im Zieloperanden abgelegt.

Exceptions im PVA-Mode:

#GP(0)	Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

OUT Ausgabe auf einen Ausgabeport **OUT**

Opkode	Befehl	Takte	Beschreibung
E6 db	OUT db,AL	3	(port db) := AL
EE	OUT DX,AL	3	(DX) := AL
E7 db	OUT db,AX	3	(port db) := AX
EF	OUT DX,AX	3	(DX) := AX

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl OUT ermöglicht die Datenausgabe des Akkumulatorinhalts (AL für Byteausgabe, AX für Wortausgabe) auf einen Ausgabeport, der durch den ersten Operanden spezifiziert wird. Portadressen können im Register DX vorgegeben werden, wobei 8-Bit-Portadressen intern durch Auffüllen mit führenden Nullen zu einem 16-Bit-Wert erweitert werden.

8-Bit-Portadressen können auch als Direktoperand angegeben werden.

Portadresse	Ein-/Ausgabe-Adreßbereich
Direktwert db	0 0FFH
Register DX	0 0FFFFH

Ausgabeports mit einer Breite von 16 Bit werden nur über gerade Portadressen angesprochen.
Die Portadressen 0F8H-0FFH sind reserviert.

Exceptions im PVA-Mode:
#GP(0) Die aktuelle Privilegierungsstufe ist höher (niedrigere Priorität) als die Stufe, die durch die Bitpositionen IOPL im Flagregister angegeben ist.

Exceptions im RA-Mode: keine

OUTS Zeichenkettenausgabe OUTS

Opkode	Befehl	Takte	Beschreibung	
6E	OUTS DX,eb	5	(DX) := [SI]	;Byteausgabe, SI setzen
6F	OUTS DX,ew	5	(DX) [SI]	;Wortausgabe, SI setzen
6E	OUTSB	5	(DX) := DS:[SI]	;Byteausgabe, SI setzen
6F	OUTSW	5	(DX) := DS:[SI]	;Wortausgabe, SI setzen

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:
Der Befehl OUTS ermöglicht die Datenausgabe aus einem fortlaufenden Speicherbereich (Zeichenkette) auf einen Ausgabeport. Die Portadresse wird im Register DX vorgegeben, während der Inhalt des Indexregisters SI die Adresse bestimmt, von der das auszugebende Element geholt wird. Die Angabe eines Segment-Override-Präfixes für den Quelloperanden ist möglich (das Segmentregister DS ist voreingestellt).
Nach der Ausgabe wird der Zeichenpointer SI in Abhängigkeit vom Flag DF (direction flag) automatisch beeinflußt:

	DF = 0	DF = 1
Byteausgabe	SI = SI + 1	SI = SI - 1
Wortausgabe	SI = SI + 2	SI = SI - 2

Portadressen sind im Bereich 0-0FFFFH zulässig (Ausnahmen: Reservierte Portadressen 0F8H-0FFH). Ausgabeports mit einer Breite von 16 Bit werden nur über gerade Portadressen angesprochen.
Zur Ausgabe von Zeichenketten kann dem Befehl OUTS der Präfix REP vorangestellt werden. Im Register CX wird dann die Anzahl der Wiederholungen vorgegeben (siehe REP).

Exceptions im PVA-Mode:

#GP(0)	Die aktuelle Privilegierungsstufe ist höher (niedrigere Priorität) als die Stufe, die durch die Bitpositionen IOPL im Flagregister angegeben ist.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

POP — Wort aus dem Stack holen — POP

Opkode	Befehl	Takte	Beschreibung
07	POP ES	5,pm=20	ES := SS:[SP] ; SP := SP + 2
17	POP SS	5,pm=20	SS := SS:[SP] ; SP := SP + 2
1F	POP DS	5,pm=20	DS := SS:[SP] ; SP := SP + 2
58 +rw	POP rw	5	rw := SS:[SP] ; SP := SP + 2
8F /0	POP mw	5	mw := SS:[SP] ; SP := SP + 2

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl POP holt ein Wort aus der Spitze des Stacks (bezeichnet durch SS:SP) und legt es auf der durch den Operanden spezifizierten Adresse (Segmentregister, Speicher, allgemeines Register) ab. Anschließend wird der Stackpointer um zwei Speicheradressen inkrementiert und zeigt damit auf die neue Spitze des Stacks.
Wurde als Operand ein Segmentregister angegeben, muß der Wert, der aus dem Stack gelesen wurde, einen gültigen Selektor darstellen. Im PVA-Mode wird zusätzlich der Inhalt des dem Selektor zugeordneten Deskriptors in der entsprechenden Systemtabelle (GDT oder LDT) in den nicht sichtbaren Teil des angegebenen Segmentregisters übertragen, wobei gleichzeitig eine Überprüfung der Gültigkeit des Deskriptoreintrags vorgenommen wird.

Wird aus dem Stack ein Nullselektor (Wortgröße im Wertebereich 0 bis 3) in eines der Segmentregister DS oder ES geladen, resultiert hieraus keine Exception. Erst wenn ein Programm versucht, über dieses Segmentregister, das den Nullselektor enthält, auf eine Speicheradresse zuzugreifen, wird eine Exception #GP(0) ausgelöst, ohne daß ein tatsächlicher Speicherzugriff erfolgte.

Bei der Ausführung des Befehls POP SS werden alle Interrupts einschließlich des nicht maskierbaren Interrupts (NMI) unterdrückt, bis dieser und der nächstfolgende Befehl vollständig abgearbeitet wurden. Somit ist es möglich, den vollständigen Stackpointer SS:SP ohne externe Unterbrechung aus dem Stack zurückzuspeichern.

Exceptions im PVA-Mode:

- if POP SS then
  - Selektor ist ungleich null else #GP(0)
  - Selektor liegt innerhalb der Grenzen der Deskriptortabelle else #GP(Selektor)
  - RPL des Selektors = CPL else #GP(Selektor)
  - Deskriptor (Accessbyte) bezeichnet ein beschreibbares Datensegment else #GP(Selektor)
  - DPL des Deskriptors = CPL else #GP(Selektor)
  - Segment ist präsent else #SS(Selektor)
  - SS := Selektor
  - SS-Cache := Segmentdeskriptor

- if POP DS oder POP ES mit Selektor ungleich null then
  - Selektor liegt innerhalb der Grenzen der Deskriptortabelle else #GP(Selektor)
  - Deskriptor (Accessbyte) beschreibt ein Datensegment oder lesbares Kodesegment else #GP(Selektor)
  - if Datensegment oder "non-conforming" Kodesegment then
    - DPL >= CPL else #GP(Selektor)
    - DPL >= RPL else #GP(Selektor)
  - Segment ist präsent else #NP(Selektor)
  - Segmentregister := Selektor
  - Segmentregister-Cache := Segmentdeskriptor

- if POP DS oder POP ES mit Nullselektor then
  - Segmentregister := Selektor
  - Segmentregister-Cache wird als ungültig markiert

#GP(0)	Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Stackpointer SP enthält vor Ausführung des Befehls den Wert OFFFFH.

POPA — POP für alle universellen Register — POPA

Opkode	Befehl	Takte	Beschreibung
61	POPA	19	POP DI,SI,BP,SP,BX,DX,CX,AX

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:
Der Befehl POPA speichert nacheinander die Inhalte der Register DI, SI, BP, SP, BX, DX, CX und AX aus dem Stack zurück. Eine Ausnahme bildet das Stackpointerregister SP, dessen Wert nicht aus dem Stack geholt wird, sondern der sich durch Addition des Stackpointerinhalts vor dem Befehl POPA mit dem Direktwert 16 ergibt (2 Byte pro Register).
POPA stellt die Umkehrung des Befehls PUSHA dar, der zur Rettung des Registersatzes benutzt werden kann.

Exceptions im PVA-Mode:
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13 Der Stackpointer SP enthält vor dem Rückspeichern eines Registers den Wert OFFFFH.

POPF Rückspeichern des Flagregisters aus dem Stack POPF

Opkode	Befehl	Takte	Beschreibung
9D	POPF	5	FLAGS := SS:[SP] ; SP := SP + 2

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	M	M	M	M	M	M	M	M	M

Operation:
Der Befehl POPF speichert das Wort, das aktuell durch den Stackpointer adressiert wird, in das Flagregister zurück. Der Stackpointer wird anschließend um den Wert 2 inkrementiert.
Die Bitpositionen IOPL (Ein-/Ausgabe-Privilegierungsstufe) werden nur geändert, wenn der Befehl in der Privilegierungsstufe 0 (höchste Privilegierung) ausgeführt wird. Ebenso wird das Flag IF (interrupt enable) nur verändert, wenn die aktuelle Privilegierungsstufe CPL mindestens der Ein-/Ausgabe-Privilegierungsstufe (IOPL) entspricht.

Achtung!
Wird der Befehl POPF nicht mit der erforderlichen Privilegierungsstufe ausgeführt, resultiert keine Exception, aber es werden auch keine privilegierten Bitpositionen (IF, IOPL) im Flagregister geändert.

Exceptions im PVA-Mode:
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13 Der Stackpointer SP enthält vor Ausführung des Befehls den Wert OFFFFH.

PUSH Wort im Stack ablegen **PUSH**

Opkode	Befehl	Takte	Beschreibung
06	PUSH ES	3	SP := SP - 2 ; SS:[SP] := ES
0E	PUSH CS	3	SP := SP - 2 ; SS:[SP] := CS
16	PUSH SS	3	SP := SP - 2 ; SS:[SP] := SS
1E	PUSH DS	3	SP := SP - 2 ; SS:[SP] := DS
50 +rw	PUSH rw	3	SP := SP - 2 ; SS:[SP] := rw
FF /6	PUSH mw	5	SP := SP - 2 ; SS:[SP] := mw
68 dw	PUSH dw	3	SP := SP - 2 ; SS:[SP] := dw
6A db	PUSH db	3	SP := SP - 2 ; SS:[SP] := db (interne Vorzeichenerweiterung auf 16 Bit)

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Bei Ausführung des Befehls PUSH wird zuerst das Stackpointerregister SP um 2 dekrementiert und anschließend der im Operandenfeld spezifizierte Wortoperand auf der nun durch den Pointer SS:SP adressierten Speicheradresse abgelegt.
Zu beachten ist, daß der Mikroprozessor 80286 bei der Ausführung des Befehls PUSH SP den Wert des Stackpointers, der vor der Befehlsausführung gültig war, im Stack ablegt (Unterschied zum Mikroprozessor 8086, der hier den neuen Stackpointerinhalt abspeichert, siehe Abschn. 15 Kompatibilität).

Exceptions im PVA-Mode:

#GP(0) Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig oder der neue Stackpointerinhalt liegt außerhalb des Stacksegmentlimits.

Exceptions im RA-Mode:

Der 80286 geht in den "shut-down"-Zustand, wenn der Stackpointer vor der Ausführung des Befehls PUSH den Wert 1 beinhaltete.

PUSHA Abspeichern der Registerinhalte im Stack **PUSHA**

Opkode	Befehl	Takte	Beschreibung
60	PUSHA	17	PUSH AX,CX,DX,BX,SP,BP,SI,DI

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:
Der Befehl PUSHA speichert nacheinander die Inhalte der Register AX, CX, DX, BX, den Wert des Stackpointerregisters SP vor der Befehlsausführung, der Register BP, SI und DI im Stack ab. Dabei wird der Wert des Stackpointers um 16 dekrementiert (entspricht den acht Wortgrößen). Mit Hilfe des Befehls POPA können zu einem späteren Zeitpunkt alle Register wieder aus dem Stack zurückgespeichert werden.

Exceptions im PVA-Mode:
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Der 80286 geht in den "shut-down"-Zustand, wenn der Wert des Stackpointers vor der Befehlsausführung 1, 3, oder 5 ist.
Enthält SP den Wert 7, 9 11, 13 oder 15, resultiert eine General-Protection-Exception.

PUSHF — Abspeichern des Flagregisters im Stack — PUSHF

Opkode	Befehl	Takte	Beschreibung
9C	PUSHF	3	SP := SP - 2 ; SS:[SP] := FLAGS

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

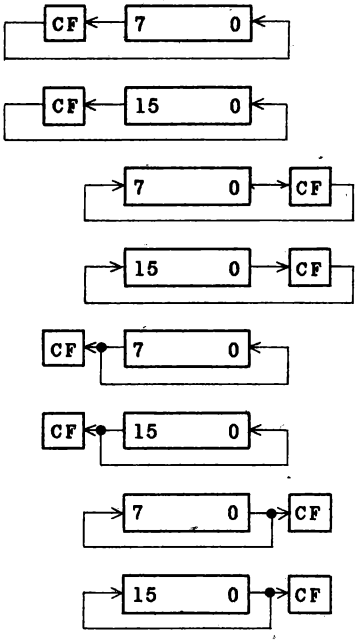
Operation:
Der Befehl PUSHF dekrementiert den Inhalt des Stackpointerregisters SP um 2 und speichert den Inhalt des Flagregisters auf der durch den Stackpointer SS:SP bezeichneten Speicheradresse ab.

Exceptions im PVA-Mode:
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Der 80286 geht in den "shut-down"-Zustand, wenn SP vor der Ausführung von PUSHF den Wert 1 beinhaltete.

RCL/RCR/ROL/ROR — Rotationsbefehle — RCL/RCR/ROL/ROR

Opkode	Befehl	Takte	Beschreibung
D0 /2	RCL eb,1	2,mem=7	Anzahl 1
D2 /2	RCL eb,CL	5,mem=8	Anzahl CL
C0 /2 db	RCL eb,db	5,mem=8	Anzahl db
D1 /2	RCL ew,1	2,mem=7	Anzahl 1
D3 /2	RCL ew,CL	5,mem=8	Anzahl CL
C1 /2 db	RCL ew,db	5,mem=8	Anzahl db
D0 /3	RCR eb,1	2,mem=7	Anzahl 1
D2 /3	RCR eb,CL	5,mem=8	Anzahl CL
C0 /3 db	RCR eb,db	5,mem=8	Anzahl db
D1 /3	RCR ew,1	2,mem=7	Anzahl 1
D3 /3	RCR ew,CL	5,mem=8	Anzahl CL
C1 /3 db	RCR ew,db	5,mem=8	Anzahl db
D0 /0	ROL eb,1	2,mem=7	Anzahl 1
D2 /0	ROL eb,CL	5,mem=8	Anzahl CL
C0 /0 db	ROL eb,db	5,mem=8	Anzahl db
D1 /0	ROL ew,1	2,mem=7	Anzahl 1
D3 /0	ROL ew,CL	5,mem=8	Anzahl CL
C1 /0 db	ROL ew,db	5,mem=8	Anzahl db
D0 /1	ROR eb,1	2,mem=7	Anzahl 1
D2 /1	ROR eb,CL	5,mem=8	Anzahl CL
C0 /1 db	ROR eb,db	5,mem=8	Anzahl db
D1 /1	ROR ew,1	2,mem=7	Anzahl 1
D3 /1	ROR ew,CL	5,mem=8	Anzahl CL
C1 /1 db	ROR ew,db	5,mem=8	Anzahl db



Flags:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
*	-	-	-	-	-	-	-	M

Bei einmaliger Rotation nach links gilt:
Das OF-Flag ergibt sich aus der Exklusiv-Oder-Verknüpfung des MSB des Resultats mit dem resultierenden CF-Flag.

Bei einmaliger Rotation nach rechts gilt:
Das OF-Flag ergibt sich aus der Exklusiv-Oder-Verknüpfung der beiden höchstwertigen Bits des Resultats.

Bei mehrfachen Rotationen (zweiter Operand ungleich eins) ist das OF-Flag undefiniert.

Operation:

Rotationsbefehle führen eine ringförmige Verschiebung der Bits des angegebenen Operanden in die im Operationskode spezifizierte Richtung aus. Im Operationskode ist ebenfalls festgelegt, ob das Flag CF in die Rotation einbezogen wird oder nicht.

Befehle RCL, RCR: Das CF-Flag wird in die bei der Rotation frei werdende Bitposition des Operanden übertragen, und das CF-Flag enthält anschließend das herausgeschobene Bit.

Befehle ROL, ROR: Das CF-Flag enthält den Wert des bei der Rotation herausgeschobenen Bits des Operanden, der gleichzeitig in die frei werdende Bitposition übertragen wird.

Der zweite Operand gibt die Anzahl der Rotationen vor (Direktwert 1, Inhalt des Byteregisters CL oder ein Direktwert).
Um die maximale Ausführungszeit der Rotationsbefehle zu begrenzen (Einhaltung bestimmter Interruptreaktionszeiten in Softwaresystemen), ist beim 80286 die maximal mögliche Anzahl der Rotationen auf 31 festgelegt (Unterschied zum Mikroprozessor 8086, siehe Abschnitt 15 Kompatibilität). Der Mikroprozessor 80286 maskiert zur Begrenzung der Anzahl der Rotationen den zweiten Operanden mit dem Wert 1FH (niederwertige fünf Bits).

Exceptions im PVA-Mode:

#GP(0)	Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von 0FFFFH.

REP — Wiederholungspräfixe — REP

Opkode	Befehl	Takte	Beschreibung
F3	REP (prefix)	5+4*CX	Wiederholpräfix für MOVS, LODS, STOS, INS, OUTS bis CX=0
F3	REPE (prefix)	5+4*CX	Wiederholpräfix für CMPS, SCAS bis ZF=0 oder CX=0
F3	REPZ (prefix)	5+4*CX	Wiederholpräfix für CMPS, SCAS bis ZF=0 oder CX=0
F2	REPNE (prefix)	5+4*CX	Wiederholpräfix für CMPS, SCAS bis ZF=1 oder CX=0
F2	REPNZ (prefix)	5+4*CX	Wiederholpräfix für CMPS, SCAS bis ZF=1 oder CX=0

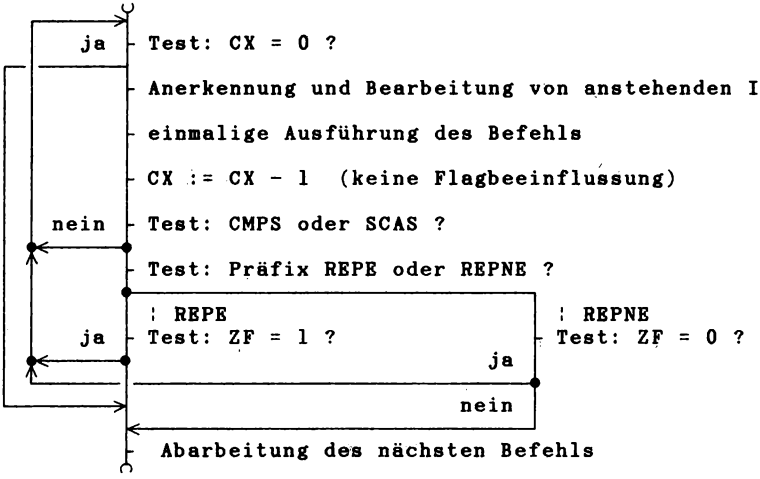
Flags:
Das Präfix REP steht nur vor Befehlen, bei denen keine Flagbeeinflussung möglich ist. Bei den Präfixen REPE und REPNE erfolgt die Beeinflussung entsprechend den Befehlen, die dem Präfix folgen.

Operation:
Der REP-Präfix bewirkt die wiederholte Ausführung des nachfolgenden Zeichenkettenmanipulationsbefehls, solange das Register CX einen Wert ungleich null enthält.
Die Präfixe REPE (repeat while equal) und REPNE (repeat while not equal), die nur in Verbindung mit den Zeichenkettenvergleichsbefehlen

CMPS und SCAS angewendet werden können, benutzen zusätzlich das Flag ZF (neben dem Inhalt des CX-Registers) als Abbruchbedingung der wiederholten Befehlsausführung. Über den Befehl JCXZ oder andere bedingte Sprungbefehle (JZ, JE, JNZ, JNE) kann unterschieden werden, ob der Abbruch durch erfolgreiche Suche oder durch das Ende der Zeichenkette hervorgerufen wurde.

Die Präfixe REPZ und REPNZ sind Synonyme für REPE und REPNE.

Die durch die REP-Präfixe ausgelösten Aktionen sind in dem nachfolgend aufgeführten Algorithmus verdeutlicht:



Die Präfixe sind immer nur auf einen nachfolgenden Zeichenkettenmanipulationsbefehl anwendbar. Wiederholungen für Befehlsgruppen sind mit Hilfe des LOOP-Befehls zu realisieren.
Bei der Ausführung der Zeichenkettenbefehle beeinflußt das Flag DF (direction flag) die Richtung, in der die Zeichenketten verarbeitet werden (DF=0: steigende Adreßrichtung; DF=1: fallende Adreßrichtung). Bei wiederholten Ein-/Ausgabe-Operationen muß gesichert sein, daß der angesprochene Ein-/Ausgabe-Port die Daten mit der erforderlichen Geschwindigkeit bereitstellen bzw. abnehmen kann.

Exceptions im PVA-Mode und RA-Mode werden in Abhängigkeit von dem verwendeten Zeichenkettenmanipulationsbefehl generiert.

RET Rückkehr aus einem Unterprogramm **RET**

Opkode	Befehl	Takte	Beschreibung
C3	RET	11	RET NEAR
CB	RET	15,pm=25	RET FAR, gleiche Privilegstufe
CB	RET	55	RET FAR, niedrigere Privilegstufe mit Stackumschaltung
C2 dw	RET dw	11	RET NEAR, SP:=SP+dw
CA dw	RET dw	15,pm=25	RET FAR, gleiche Privilegstufe, SP:=SP+dw
CA dw	RET dw	55	RET FAR, niedrigere Privilegstufe mit Stackumschaltung, SP:=SP+dw

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl RET bewirkt den Austritt aus einem Unterprogramm. Die Rückkehradresse, die beim Eintritt in das Unterprogramm (Befehl CALL) im Stack abgespeichert wurde, wird zurückgeladen, und die Befehlsabarbeitung wird fortgesetzt.

Der optionale numerische Parameter dient zur Freigabe von Stackdatenspeicher, der beim Unterprogrammaufruf zur Parameterübergabe belegt wurde. Er spezifiziert die Byteanzahl, die für die Parameterübergabe benötigt wird. Der Wert wird nach dem Rückspeichern der Register zum Stackpointerregister SP addiert.

Erfolgt die Rückkehr in das aufrufende Programm im gleichen Programmkodesegment (RET NEAR), wird nur der Inhalt des IP-Registers zurückgespeichert (CS bleibt unverändert).

Bei der Rückkehr in ein anderes Programmkodesegment (RET FAR) wird der im Stack abgelegte 4-Byte-Pointer in die Register CS:IP zurückgeladen (erst Offset, anschließend Selektor).

Im PVA-Mode wird der Segmentdeskriptor des Selektors ausgewertet. Er muß ein Programmkodesegment gleicher oder niedrigerer Privilegierung (DPL größer oder gleich CPL) bezeichnen. Ist die Privilegierung des Deskriptors der Rückkehradresse niedriger als die des Unterprogramms, wird der Stackpointer (SS:SP), der beim Unterprogrammaufruf mit Wechsel der Privilegierungsstufe vor der Rückkehradresse im Stack abgespeichert wurde, zurückgeladen.

Bei einer Rückkehr in ein Programm mit niedrigerer Privilegierungsstufe kann der Zustand eintreten, daß die Segmentregister DS oder ES auf Segmente zeigen, für die in der niedrigen Privilegierungsstufe keine Zugriffsberechtigung mehr besteht. In diesem Fall setzt die 80286-CPU diese Segmentregister automatisch zu Null, um nichtautorisierte Datenzugriffe zu verhindern. Zusätzlich wird eine General-Protection-Exception generiert.

```
Exceptions im PVA-Mode:

- 2 Worte oberhalb des Stackpointers liegen innerhalb des Stacksegment-
  limits else #SS(0)
- RPL des Rückkehrselektors >= CPL else #GP(Rückkehrselektor)
- if RPL des Rückkehrselektors = CPL then

   Rückkehr zu gleicher Privilegierung

   - Rückkehrselektor ist ungleich null else #GP(0)
   - Rückkehrselektor liegt innerhalb der Grenzen der Deskriptortabelle
     else #GP(Rückkehrselektor)
   - Deskriptor (Accessbyte) beschreibt ein Kodesegment else #GP(Rückkehr-
     selektor)
   - if Kodesegment "non-conforming" then
      - DPL des Kodesegments = CPL else #GP(Rückkehrselektor)
   - if Kodesegment "conforming" then
      - DPL des Kodesegments <= CPL else #GP(Rückkehrselektor)
   - Programmkodesegment ist präsent else #NP(Rückkehrselektor)
   - das oberste Wort im Stack liegt innerhalb des Stacksegmentlimits
     else #SS(0)
   - IP liegt innerhalb des Kodesegmentlimits else #GP(0)
   - CS:IP := Stack
   - CS-Cache := Deskriptor
   - SP := SP + 4 + Direktoperand (falls vorhanden)

ELSE

   - Rückkehr zu niedrigerer Privilegierung

   - 8 Worte + Direktoperand oberhalb des Stackpointers liegen innerhalb
     des Stacksegmentlimits else #SS(0)
   - Test Rückkehrselektor (SP+2) und zugehöriger Deskriptor:
      - Rückkehrselektor ist ungleich null else #GP(0)
      - Rückkehrselektor liegt innerhalb der Grenzen der Deskriptortabelle
        else #GP(Rückkehrselektor)
   - Deskriptor (Accessbyte) beschreibt ein Kodesegment else #GP(Rückkehr-
     selektor)
      - if Kodesegment "non-conforming" then
         - DPL des Kodesegments = RPL des Rückkehrselektors else #GP(Rück-
           kehrselektor)
      - if Kodesegment "conforming" then
         - DPL des Kodesegments <= RPL des Rückkehrselektors else
           #GP(Rückkehrselektor)
      - Programmkodesegment ist präsent else #NP(Rückkehrselektor)

   - Test Selektor SS für Rückkehr (SP+6+Direktoperand) und zugehöriger
     Deskriptor:
      - Selektor ist ungleich null else #GP(0)
      - Selektor liegt innerhalb der Grenzen der Deskriptortabelle else
        #GP(SS-Selektor)
      - RPL des Stackselektors = RPL des Rückkehrkodesegmentselektors else
        #GP(SS-Selektor)
```

- Deskriptor (Accessbyte) bezeichnet ein beschreibbares Datensegment else #GP(SS-Selektor)
- DPL des Stacksegments = RPL des Rückkehrkodesegments else #GP(SS-Selektor)
- Stacksegment ist präsent else #NP(SS-Selektor)

- IP liegt innerhalb des Kodesegmentlimits else #GP(0)
- CPL := RPL des Rückkehrkodesegmentselektors CS
- CS:IP := Stack
- RPL-Feld des CS-Registers := CPL
- SP := SP + 4 + Direktoperand (falls vorhanden)
- SS:SP := Stack
- CS-Cache := Deskriptor
- SS-Cache := Deskriptor

Für ES und DS gilt:

- if Segmentregisterinhalt ist in der neuen Privilegierungsstufe nicht zulässig then
  - Segmentregister (Selektor und Accessbyte) := 0

- Segmentregisterinhalt ist zulässig, wenn gilt:
  - Selektor liegt innerhalb der Grenzen der Deskriptortabelle
  - Deskriptor (Accessbyte) beschreibt ein Datensegment oder ein lesbares Kodesegment
  - if Datensegment oder Kodesegment "non-conforming" then
    - DPL >= CPL
    - DPL >= RPL

Exceptions im RA-Mode:
Interrupt 13 Die Offsetkomponente des Stackpointers steht vor der Ausführung des RET-Befehls auf OFFFFH

SAHF — Übertragen der Werte aus AH in das Flagregister — SAHF

Opkode	Befehl	Takte	Beschreibung
9E	SAHF	2	FLAGS := AH (AH = SF ZF xx AF xx PF xx CF)

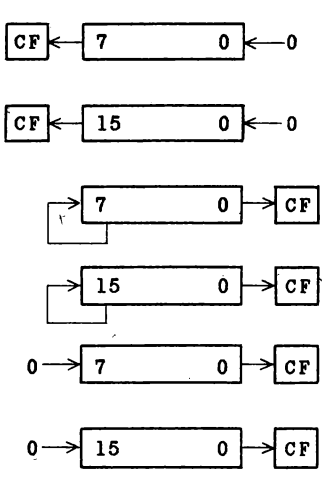
Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	M	M	M	M	M

Operation:
SAHF überträgt die angegebenen Flags aus dem Register AH in das Flagregister. Wie oben ersichtlich, werden nur die Flags übertragen, die von den arithmetischen Operationen beeinflußt werden (vgl. Befehl LAHF).

Keine Exceptions.

SAL/SAR/SHL/SHR Schiebebefehle **SAL/SAR/SHL/SHR**

Opkode	Befehl	Takte	Beschreibung
D0 /4	SAL eb,1	2,mem=7	Anzahl 1
D2 /4	SAL eb,CL	5,mem=8	Anzahl in CL
C0 /4 db	SAL eb,db	5,mem=8	Anzahl db
D1 /4	SAL ew,1	2,mem=7	Anzahl 1
D3 /4	SAL ew,CL	5,mem=8	Anzahl in CL
C1 /4 db	SAL ew,db	5,mem=8	Anzahl db
D0 /7	SAR eb,1	2,mem=7	Anzahl 1
D2 /7	SAR eb,CL	5,mem=8	Anzahl in CL
C0 /7 db	SAR eb,db	5,mem=8	Anzahl db
D1 /7	SAR ew,1	2,mem=7	Anzahl 1
D3 /7	SAR ew,CL	5,mem=8	Anzahl in CL
C1 /7 db	SAR ew,db	5,mem=8	Anzahl db
D0 /5	SHR eb,1	2,mem=7	Anzahl 1
D2 /5	SHR eb,CL	5,mem=8	Anzahl in CL
C0 /5 db	SHR eb,db	5,mem=8	Anzahl db
D1 /5	SHR ew,1	2,mem=7	Anzahl 1
D3 /5	SHR ew,CL	5,mem=8	Anzahl in CL
C1 /5 db	SHR ew,db	5,mem=8	Anzahl db



Flags: OF DF IF TF SF ZF AF PF CF
* - - - M M U M M

Bei einmaliger Linksverschiebung gilt:
Das OF-Flag ergibt sich aus der Exklusiv-Oder-Verknüpfung des MSB des Resultats mit dem resultierenden CF-Flag.

Bei einmaliger arithmetischer Rechtsverschiebung (SAR) gilt:
OF := 0

Bei einmaliger logischer Rechtsverschiebung (SHR) gilt:
OF := MSB(Operand vor der Befehlsausführung)

Bei Verschiebungen um mehrere Bitpositionen (zweiter Operand ungleich eins) ist das OF-Flag undefiniert.

Operation:

Der Befehl SAL (shift arithmetic left) führt eine Linksverschiebung des Operanden aus, wobei das höchstwertige Bit in das CF-Flag übertragen wird und die niederwertigste Bitposition mit einer Null aufgefüllt wird. Der Befehl SAL ist identisch mit dem Befehl SHL (shift left).
SAR und SHR führen Rechtsverschiebungen des Operanden aus. Dabei wird das niederwertigste Bit in das CF-Flag übertragen. Bei Ausführung des Befehls SAR (shift arithmetic right) bleibt das Vorzeichen erhalten (entspricht einer vorzeichenbehafteten Division durch 2), während beim Befehl SHR (shift right) die frei werdende höchste Bitposition mit einer Null aufgefüllt wird (vorzeichenlose Division durch 2).

Der zweite Operand gibt die Anzahl der Verschiebungen vor (Direktwert 1, Inhalt des Byteregisters CL oder ein Direktwert).
Um die maximale Ausführungszeit der Verschiebebefehle zu begrenzen (Einhaltung definierter Interruptreaktionszeiten), ist beim 80286 die maximal mögliche Anzahl der Verschiebungen auf 31 festgelegt (Unterschied zum Mikroprozessor 8086, siehe Abschnitt 15. Kompatibilität). Der Mikroprozessor 80286 maskiert zur Begrenzung der Anzahl der Verschiebungen den zweiten Operanden mit dem Wert 1FH (niederwertige fünf Bits).

Exceptions im PVA-Mode:

#GP(0)	Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

SBB/SUB — Subtraktion von Integerzahlen — SBB/SUB

Opkode	Befehl	Takte	Beschreibung
18 /r	SBB eb,rb	2,mem=7	eb = eb - rb - CF
19 /r	SBB ew,rw	2,mem=7	ew = ew - rw - CF
1A /r	SBB rb,eb	2,mem=7	rb = rb - eb - CF
1B /r	SBB rw,ew	2,mem=7	rw = rw - ew - CF
1C db	SBB AL,db	3	AL = AL - db - CF
1D dw	SBB AX,dw	3	AX = AX - dw - CF
80 /3 db	SBB eb,db	3,mem=7	eb = eb - db - CF
81 /3 dw	SBB ew,dw	3,mem=7	ew = ew - dw - CF
83 /3 db	SBB ew,db	3,mem=7	ew = ew - db - CF
28 /r	SUB eb,rb	2,mem=7	eb = eb - rb
29 /r	SUB ew,rw	2,mem=7	ew = ew - rw
2A /r	SUB rb,eb	2,mem=7	rb = rb - eb
2B /r	SUB rw,ew	2,mem=7	rw = rw - ew
2C db	SUB AL,db	3	AL = AL - db
2D dw	SUB AX,dw	3	AX = AX - dw
80 /5 db	SUB eb,db	3,mem=7	eb = eb - db
81 /5 dw	SUB ew,dw	3,mem=7	ew = ew - dw
83 /5 db	SUB ew,db	3,mem=7	ew = ew - db

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	M	-	-	-	M	M	M	M	M

Operation:

Mit Hilfe der Befehle SBB und SUB können zwei Integerzahlen subtrahiert werden. Der Befehl SBB subtrahiert einen vorhandenen Übertrag (angegeben durch den Wert des Flags CF vor der Befehlsausführung) vom Ergebnis. Das Ergebnis wird im ersten Operanden abgelegt. SBB wird

meist bei der Subtraktion von Integeroperanden benutzt, die aus mehreren Worten bestehen.
Direkt angegebene Byteoperanden, die von Wortoperanden subtrahiert werden sollen, werden intern vor der Subtraktion vorzeichenbehaftet auf Wortgröße erweitert.

Exceptions im PVA-Mode:
#GP(0) Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13 Der Wortoperand besitzt ein Offset von OFFFFH.

SCAS Suchen von Elementen in Zeichenketten SCAS

Opkode	Befehl	Takte	Beschreibung
AE	SCAS mb	7	Bytesuche AL - ES:[DI] ; DI setzen
AF	SCAS mw	7	Wortsuche AX - ES:[DI] ; DI setzen
AE	SCASB	7	Bytesuche AL - ES:[DI] ; DI setzen
AF	SCASW	7	Wortsuche AX - ES:[DI] ; DI setzen

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	M	-	-	-	M	M	M	M	M

Operation:
Der Zeichenkettenbefehl SCAS dient zur Suche ausgewählter Elemente (8 Bit oder 16 Bit) in Zeichenketten durch Subtraktion vom Akkumulatorinhalt. Der Befehl beeinflußt nur die Flags, die für nachfolgende bedingte Sprungbefehle gesetzt werden. Die Operanden und der Akkumulator bleiben unverändert. Der Operand muß immer über das Registerpaar ES:DI adressiert werden (Segment-Override-Präfix ist nicht anwendbar).
Nach dem Vergleich wird das Register DI in Abhängigkeit vom Flag DF (direction flag) wie folgt verändert:

	DF = 0	DF = 1
Bytesuche	DI = DI + 1	DI = DI - 1
Wortsuche	DI = DI + 2	DI = DI - 2

Zur Suche in Speicherblöcken können den Befehlen SCAS die Präfixe REPE oder REPNE vorangestellt werden. Im Register CX wird die Anzahl der Wiederholungen vorgegeben (siehe Präfix REP).

```
Exceptions im PVA-Mode:
#GP(0)          Die effektive Adresse eines Speicheroperanden in einem der
                Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)          Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13    Der Wortoperand besitzt ein Offset von OFFFFH.
```

SGDT/SIDT — Rückspeichern der Register GDTR/IDTR — SGDT/SIDT

Opkode	Befehl	Takte	Beschreibung
0F 01 /0	SGDT m	11	m m+5 := GDTR
0F 01 /1	SIDT m	12	m ... m+5 := IDTR

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Die Inhalte des Globaldeskriptortabellenregisters (GDTR) bzw. des Interruptdeskriptortabellenregisters (IDTR) werden, beginnend bei der angegebenen Adresse, im Speicher abgelegt. Der sechs Byte große Speicherblock hat nach der Befehlsausführung folgenden Inhalt:

m+0: Länge der Deskriptortabelle (Bit 7-0)
m+1: Länge der Deskriptortabelle (Bit 15-8)
m+2: Basisadresse der Deskriptortabelle (Bit 7-0)
m+3: Basisadresse der Deskriptortabelle (Bit 15-8)
m+4: Basisadresse der Deskriptortabelle (Bit 23-16)
m+5: undefiniert

Die Befehle SGDT und SIDT werden nur in Betriebssystemprogrammen benutzt (vgl. Befehle LGDT und LIDT).

```
Exceptions im PVA-Mode:
#UD             Der Quelloperand ist ein Register.
#GP(0)          Der Zieloperand befindet sich in einem schreibgeschützten
                Segment.
#GP(0)          Die effektive Adresse eines Speicheroperanden in einem der
                Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)          Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
                Diese Befehle sind im RA-Mode gültig, um die Initialisie-
                rungen zum Übergang in den PVA-Mode zu überprüfen.

Interrupt 13    Der Wortoperand besitzt ein Offset von OFFFFH.
#UD             Der Quelloperand ist ein Register.
```

SLDT Rückspeichern des Registers LDTR **SLDT**

Opkode	Befehl	Takte	Beschreibung
0F 00 /0	SLDT ew	2,mem=3	ew := LDTR

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl SLDT legt den Inhalt des Lokaldeskriptortablellenregisters (LDTR) auf der angegebenen effektiven Adresse ab. Diese Adresse enthält anschließend einen Selektor, der auf einen LDT-Deskriptor in der GDT zeigen muß.
Dieser Befehl erscheint nur in Betriebssystemprogrammen (vgl. Befehl LLDT).

Exceptions im PVA-Mode:

#GP(0)	Der Zieloperand befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 6	Der Befehl SLDT ist im RA-Mode nicht zulässig.

SMSW Rückspeichern des Maschinenstatusworts **SMSW**

Opkode	Befehl	Takte	Beschreibung
0F 01 /4	SMSW ew	2,mem=3	ew := MSW

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Inhalt des Maschinenstatusworts wird auf der angegebenen effektiven Adresse abgelegt (vgl. Befehl LMSW).

Exceptions im PVA-Mode:

#GP(0)	Der Zieloperand befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

STC Setzen des Flags CF **STC**

Opkode	Befehl	Takte	Beschreibung
F9	STC	2	CF := 1

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	1

Operation:
Der Befehl STC setzt das Flag CF (carry flag) im Flagregister der 80286-CPU auf den Wert 1 (vgl. Befehle CLC und CMC).

Keine Exceptions.

STD Setzen des Flags DF **STD**

Opkode	Befehl	Takte	Beschreibung
FD	STD	2	DF := 1

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	1	-	-	-	-	-	-	-

Operation:
Bei Ausführung des Befehls STD wird das Flag DF (direction flag) in der 80286-CPU gesetzt. Das Flag DF bestimmt die Richtung für das automatische Weitersetzen der Pointer (Indexregister SI bzw. DI) bei der Ausführung der Zeichenkettenbefehle des Mikroprozessors 80286. Nach Ausführung des Befehls STD (DF=1) werden Zeichenketten in fallender Adreßrichtung verarbeitet (vgl. Befehl CLD).

Keine Exceptions.

STI Setzen des Interruptflags **STI**

Opkode	Befehl	Takte	Beschreibung
FB	STI	2	IF := 1

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	1	-	-	-	-	-	-

Operation:
Das Interruptflag IF im Flagregister wird gesetzt, wenn die aktuelle Privilegierungsstufe mindestens der Ein-/Ausgabe-Privilegierungsstufe IOPL entspricht (Bitpositionen IOPL im Flagregister). Externe Interruptanforderungen werden nach der Befehlsabarbeitung des nächsten Befehls wieder zugelassen (sofern der nächste Befehl das IF-

Flag nicht wieder rücksetzt). Somit ist es möglich, vor dem Verlassen eines Unterprogramms, das durch externe Interrupts nicht unterbrechbar sein soll, die Interruptsperre aufzuheben und das Unterprogramm zu verlassen, ohne daß zwischen Interruptfreigabe und Rückkehrbefehl RET ein Interrupt anerkannt wird.

```
STI
RET
```

Ebenso bewirkt die Befehlsfolge

```
STI
CLI
```

keine kurzzeitige Interruptfreigabe (vgl. Befehl CLI).

Exceptions im PVA-Mode:
#GP(0) Die aktuelle Privilegierungsstufe CPL ist größer (niedrigerer Vorrang) als die Stufe, die durch die Bitpositionen IOPL eingestellt ist.

Exceptions im RA-Mode: keine

STOS — Laden von Zeichenketten — STOS

Opkode	Befehl	Takte	Beschreibung
AA	STOS mb	3	ES:[DI] := AL ; DI setzen
AB	STOS mw	3	ES:[DI] := AX ; DI setzen
AA	STOSB	3	ES:[DI] := AL ; DI setzen
AB	STOSW	3	ES:[DI] := AX ; DI setzen

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl STOS dient zum Initialisieren von Zeichenketten. Der Inhalt des Akkumulators (AL bei Byteoperanden bzw. AX bei Wortoperanden) wird in die durch den Pointer ES:DI adressierte Speicherzelle geladen.

Der Zieloperand muß über das Segmentregister ES adressiert werden. Die Angabe eines Segment-Override-Präfix ist nicht möglich.

Nach dem Datentransfer wird das Register DI automatisch so gesetzt, daß jeweils das nächste Element (Byte bzw. Wort) der Zeichenkette adressiert wird.

Die Adreßrichtung, in der die Zeichenkette behandelt wird, ist durch das Flag DF (direction flag) vorgegeben.

	DF=0	DF=1
STOSB	DI := DI + 1	DI := DI - 1
STOSW	DI := DI + 2	DI := DI - 2

Mit Hilfe des Präfixes REP kann mit einem Befehl eine Zeichenkette der Länge, die durch das Register CX vorgegeben ist (maximal 64 KByte), mit dem Akkumulatorinhalt initialisiert werden.

Exceptions im PVA-Mode:
#GP(0) Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13 Der Wortoperand besitzt ein Offset von 0FFFFH.

STR Rückspeichern des Taskregisters **STR**

Opkode	Befehl	Takte	Beschreibung
0F 00 /1	STR ew	2,mem=3	ew := TR

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:
Mit dem Befehl STR wird der Inhalt des Taskregisters der 80286-CPU unter der durch den zweiten Operanden angegebenen effektiven Adresse abgelegt (vgl. Befehl LTR).

Exceptions im PVA-Mode:
#GP(0) Der Zieloperand befindet sich in einem schreibgeschützten Segment.
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 6 Der Befehl STR ist im RA-Mode nicht zulässig.

TEST Logischer Vergleich **TEST**

Opkode	Befehl	Takte	Beschreibung
84 /r	TEST eb,rb	2,mem=6	eb UND rb ?
84 /r	TEST ew,rw	2,mem=6	ew UND rw ?
85 /r	TEST rb,eb	2,mem=6	rb UND eb ?
85 /r	TEST rw,ew	2,mem=6	rw UND ew ?
A8 db	TEST AL,db	3	AL UND db ?
A9 dw	TEST AX,dw	3	AX UND dw ?
F6 /0 db	TEST eb,db	3,mem=6	eb UND db ?
F7 /0 dw	TEST ew,dw	3,mem=6	ew UND dw ?

Flag:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	0	-	-	-	M	M	U	M	0

Operation:

Der Befehl TEST führt einen logischen Vergleich zweier Operanden über eine logische UND-Verknüpfung (siehe Befehl AND) durch. Im Gegensatz zum Befehl AND wird das Resultat der UND-Verknüpfung nicht über den ersten Operanden übergeben, sondern es werden lediglich die Flags gesetzt. Sie können über nachfolgende bedingte Sprungbefehle für Programmverzweigungen genutzt werden.

Exceptions im PVA-Mode:

#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von OFFFFH.

VERR/VERW Test auf Lese/Schreib-Erlaubnis für ein Segment **VERR/VERW**

Opkode	Befehl	Takte	Beschreibung
0F 00 /4	VERR ew	14,mem=16	ZF=1, wenn Segment ew lesbar
0F 00 /5	VERW ew	14,mem=16	ZF=1, wenn Segment ew beschreibbar

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	M	-	-	-

Operation:

Die Befehle VERR und VERW dienen zum Überprüfen, ob von der aktuellen Privilegierungsstufe aus der Zugriff auf ein spezifiziertes Speichersegment für Leseoperationen (VERR) bzw. für Schreiboperationen (VERW) erlaubt ist. Der Operand muß dabei einen Selektor enthalten, der das gewünschte Segment bezeichnet. Das Resultat der Überprüfung wird im Flag ZF übergeben (ZF=1: Zugriff erlaubt).

Im einzelnen werden folgende Tests durchgeführt:

1. Der Selektor muß auf einen definierten Segmentdeskriptor innerhalb der Grenzen einer der Deskriptortabellen (GDT oder LDT) zeigen.

2. Der durch den Selektor spezifizierte Deskriptor muß ein Programmkode- oder Datensegment bezeichnen.

3. Bei Ausführung des Befehls VERR muß das Segment lesbar sein. Analog gilt für den Befehl VERW, daß für das Segment Schreiberlaubnis vorliegen muß.

4. Beschreibt der Deskriptor ein Programmkodesegment (non-conforming), muß die aktuelle Privilegierung größer oder gleich der Deskriptorprivilegierung sein, d.h., es muß gelten:

   DPL >= CPL, RPL

   Für "conforming"-Programmkodesegmente gilt diese Bedingung nicht.

Die Überprüfungen entsprechen denen, die immer dann durchgeführt werden, wenn ein Segmentdeskriptor in eines der Segmentregister DS oder ES geladen wurde und der Zugriff auf dieses Segment erfolgt. Der Unterschied besteht darin, daß bei Erkennung einer Zugriffsverletzung keine Exception ausgelöst wird, sondern nur das Flag ZF den Wert null enthält. In Softwaresystemen kann über diese Befehle vor dem eigentlichen Datenzugriff die Zugriffserlaubnis getestet werden.

Exceptions im PVA-Mode:

#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 6	Die Befehle sind im RA-Mode nicht zulässig.

WAIT — Warten, bis /BUSY-Pin der CPU inaktiv ist — WAIT

Opkode	Befehl	Takte	Beschreibung
9B	WAIT	3	Warten, bis /BUSY-Pin inaktiv ist (HIGH)

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl WAIT dient zur Synchronisation eines Programms mit einem Numerikcoprozessor. Der Mikroprozessor 80286 wird in einen Wartezustand gebracht, bis das Eingangssignalpin /BUSY an der CPU inaktiv (HIGH) ist (numerische Operation ist beendet). Dieses Signalpin ist normalerweise mit einem Numerikcoprozessor verbunden, der dieses

Signal zu Beginn der Ausführung numerischer Befehle aktiviert (LOW).

Exceptions im PVA-Mode und RA-Mode:
#NM Das TS-Bit im MSW ist gesetzt.
#MF Der Numerikcoprozessor hat einen nicht maskierten numerischen Fehler gemeldet.

XCHG — Austausch Register/Speicher mit Register — XCHG

Opkode	Befehl	Takte	Beschreibung
86 /r	XCHG eb,rb	3,mem=5	Austausch eb mit rb
86 /r	XCHG rb,eb		
87 /r	XCHG ew,rw	3,mem=5	Austausch ew mit rw
87 /r	XCHG rw,ew		
90 +rw	XCHG AX,rw	3	Austausch AX mit rw
90 +rw	XCHG rw,AX		

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:
Der Befehl XCHG vertauscht beide angegebenen Operanden. Die Reihenfolge der Angabe der Operanden im Befehl ist nicht entscheidend. Während des Vertauschens aktiviert die 80286-CPU automatisch das Bussignal /LOCK, unabhängig davon, ob das Präfix LOCK angegeben wurde oder nicht. Auch die Ein-/Ausgabe-Privilegierung (IOPL-Bits im Flagregister) hat keinen Einfluß auf die Aktivierung des Signals /LOCK.

Exceptions im PVA-Mode:
#GP(0) Ein Operand befindet sich in einem schreibgeschützten Segment.
#GP(0) Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0) Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:
Interrupt 13 Der Wortoperand besitzt ein Offset von OFFFFH.

XLAT — Tabellenzugriff — XLAT

Opkode	Befehl	Takte	Beschreibung
D7	XLAT mb	5	AL := DS:[BX + AL]
D7	XLATB	5	AL := DS:[BX + AL]

Flags:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Operation:

Der Befehl XLAT dient zum Auffinden definierter Byteelemente in geordneten Tabellen. Die Startadresse der Tabelle wird vor der Befehlsausführung im Register BX vorgegeben. Im Byteregister AL muß der Versatz des gesuchten Tabellenelements enthalten sein. XLAT überträgt das so bezeichnete Tabellenelement in das Byteregister AL.
In der ersten Form des Befehls besteht die Möglichkeit der Angabe eines Segment-Override-Präfixes, während in der Form XLATB das voreingestellte Segmentregister DS benutzt wird.

Exceptions im PVA-Mode:

#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von 0FFFFH.

XOR — Logisches exklusives ODER — XOR

Opkode	Befehl	Takte	Beschreibung
30 /r	XOR eb,rb	2,mem=7	eb := eb XOR rb
31 /r	XOR ew,rw	2,mem=7	ew := ew XOR rw
32 /r	XOR rb,eb	2,mem=7	rb := rb XOR eb
33 /r	XOR rw,ew	2,mem=7	rw := rw XOR ew
34 db	XOR AL,db	3	AL := AL XOR db
35 dw	XOR AX,dw	3	AX := AX XOR dw
80 /6 db	XOR eb,db	3,mem=7	eb := eb XOR db
81 /6 dw	XOR ew,dw	3,mem=7	ew := ew XOR dw

Flag:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	0	-	-	-	M	M	U	M	0

Operation:

Der Befehl realisiert eine bitweise logische exklusive ODER-Verknüpfung (XOR) der Operanden. Sind korrespondierende Bitpositionen beider Operanden gleich, wird das Resultatbit auf den Wert 0 gesetzt; sind sie verschieden, erhält das Resultatbit den Wert 1. Das Resultat wird im Zieloperanden abgelegt.

Exceptions im PVA-Mode:

#GP(0)	Das Resultat befindet sich in einem schreibgeschützten Segment.
#GP(0)	Die effektive Adresse eines Speicheroperanden in einem der Segmente CS, DS oder ES ist nicht zulässig.
#SS(0)	Die Adresse im Segment SS ist nicht zulässig.

Exceptions im RA-Mode:

Interrupt 13	Der Wortoperand besitzt ein Offset von 0FFFFH.

14. Systeminitialisierung

Nach dem Zuschalten der Betriebsspannung oder nach einem RESET enthalten die Register des Mikroprozessors 80286 vordefinierte Werte.

Tafel 14.1. Voreingestellte Registerinhalte nach einem RESET

Register		Wert
Flagregister	F	0002H
Maschinenstatuswort	MSW	FFF0H
Instruktionspointer	IP	FFF0H
CS-Selektor	CS	F000H
DS-Selektor	DS	0000H
ES-Selektor	ES	0000H
SS-Selektor	SS	0000H
CS-Basisadresse		FF0000H
DS-Basisadresse		000000H
ES-Basisadresse		000000H
SS-Basisadresse		000000H
CS-Länge		FFFFH
DS-Länge		FFFFH
ES-Länge		FFFFH
SS-Länge		FFFFH
IDT-Basisadresse		000000H
IDT-Länge		03FFH

Wie aus dem Inhalt des Maschinenstatusworts (MSW) ersichtlich, beginnt der Mikroprozessor 80286 die Befehlsausführung nach dem RESET im REAL ADDRESS MODE (das Bit 'protected mode enable' ist null: PE=0).
Ein Numerikcoprozessor wird weder vorausgesetzt noch emuliert (Maschinenstatusbits MP=EM=0).

Die Startadresse, von der der erste Befehl gelesen wird, ergibt sich im RA-Mode aus dem Registerpaar CS:IP wie folgt (Abschn. 5. Adreßraum):

```
 F0000H        (Segmentbasisadresse)
+0FFF0H        (Offset)
 FFFF0H
```

Auf dieser Adresse sollte ein absoluter Sprung zur eigentlichen Initialisierungsroutine des Softwaresystems eingetragen werden.
Da im RA-Mode des 80286 der Adreßraum auf 1 MByte begrenzt ist, werden in dieser Betriebsart die vier höchstwertigen Adreßbits (Bit 20 bis 23) nicht benutzt. Diese Adreßleitungen sind jedoch nach RESET auf den Wert 1 gesetzt, so daß das letzte 64 KByte große Speichersegment aus dem 16-MByte-Adreßraum angesprochen wird. Dieser Zustand ist durch die Besonderheit gekennzeichnet, daß, obwohl sich der Pozessor im RA-Mode befindet, der Zugriff auf Speicheradressen oberhalb 1 MByte erfolgt. Dieser Zustand wird jedoch beim ersten Sprungbefehl, der das Segmentregister CS verändert

(JMP FAR), aufgehoben (Adreßleitungen 20 bis 23 werden dann rückgesetzt). Festgelegt ist auch die Lage der Interruptdeskriptortabelle (IDT) nach dem Systemstart, die sich ab Adresse 0 im Speicher befindet. Die Länge der IDT ist so bemessen, daß 256 Deskriptoreinträge aufgenommen werden können.

Die Segmentregister DS, ES und SS sind so voreingestellt, daß sie den Zugriff auf das erste 64 KByte große Speichersegment gestatten (analog 8086). Das Flagregister beinhaltet in allen definierten Bitpositionen den Wert 0. Damit ist auch sichergestellt, daß nach einem RESET keine maskierbaren Interrupts zugelassen sind (IF=0).

Die Initialisierung eines Softwaresystems, das im PVA-Mode arbeiten soll, kann entweder zu großen Teilen im RA-Mode oder fast vollständig im PVA-Mode erfolgen. Zu beachten ist hierbei, daß im RA-Mode nur das erste Megabyte des Speichers adressierbar ist. Der Zugriff auf den gesamten 16 MByte großen Adreßraum ist nur im PVA-Mode möglich.
Für eine Initialisierung im PVA-Mode spricht der Faktor, daß die Schutzmechanismen des 80286 erst in dieser Betriebsart wirksam sind. Es ist bei der Programmierung allerdings darauf zu achten, daß die Eigenschaften des PVA-Mode immer erst dann benutzt werden können, wenn die speziell für diese Eigenschaft notwendigen Initialisierungen bereits vollständig realisiert sind.

14.1. Initialisierung im RA-Mode

Die Systeminitialisierung für ein Softwaresystem, das im RA-Mode arbeitet, kann in folgendem Algorithmus zusammengefaßt werden:

1. Zuweisung des Stackbereichs (SS:SP).
2. Laden der Programme und Daten in den Speicher.
3. Programmieren der Peripheriebausteine.
4. Laden der Interrupttabelle.
5. Laden des Flagregisters (u.a. Freigabe maskierbarer Interrupts) und des MSW.
6. Start des Hauptprogramms.

Die Problematik der Systeminitialisierung im RA-Mode soll an dieser Stelle nicht ausführlich behandelt werden. Hier sei auf /8/ verwiesen.

14.2. Initialisierung im PVA-Mode

Die Umschaltung vom RA-Mode in den PVA-Mode erfolgt durch Setzen des PE-Bits im Maschinenstatuswort (mit Hilfe des Befehls LMSW). Nach dem Umschalten befindet sich der Prozessor in der Privilegierungsstufe 0 (höchste Privilegierung). Durch die Umschaltung in den PVA-Mode werden die Segmentregister nicht verändert, d.h., sie zeigen auf die gleichen Speicheradressen wie vor der Umschaltung im RA-Mode.
Sofort nach der Umschaltung in den PVA-Mode muß ein Sprung mit kurzer Distanz (JMP NEAR oder JMP SHORT) folgen, um die interne Befehlswarteschlange des Mikroprozessors zu leeren. Dieser Befehl ist notwendig, weil Befehlskodierungen, die noch im RA-Mode in diese Befehlswarteschlange

aufgenommen wurden, im PVA-Mode anders interpretiert werden können.
Obwohl nach einem RESET maskierbare Interrupts nicht erlaubt sind, müssen zu einem möglichst frühen Zeitpunkt einige Initialisierungen für das Interruptssystem vorgenommen werden. Da beim Systemstart meist noch keine Behandlungsroutinen für nichtmaskierbare Interrupts (NMI) und Exceptions abarbeitungsbereit sind, bietet es sich an, für diese Fälle den Prozessor in den "shut-down"-Zustand zu bringen. Dazu ist das Interruptdeskriptortabellenregister (IDTR) mit einem Nulldeskriptor zu laden. Wird unter dieser Bedingung eine Exception oder ein nichtmaskierbarer Interrupt (NMI) ausgelöst, findet der Prozessor im PVA-Mode keine gültige Startadresse für eine Behandlungsroutine. Dieser Doppelfehler bewirkt den Übergang in den "shut-down"-Zustand. Zu einem späteren Zeitpunkt, nachdem die Voraussetzungen zur Abarbeitung der Behandlungsroutinen geschaffen wurden, kann das Register IDTR mit der aktuellen Interruptdeskriptortabelle geladen werden. Die Interruptfreigabe sollte auch erst dann erfolgen.

Vor dem ersten Zugriff auf das Stacksegment ist das Stacksegmentregister - unabhängig von der Betriebsart - mit einem Deskriptor zu laden, der auf einen RAM-Speicherbereich zeigt. Wurde das Stacksegmentregister im RA-Mode geladen, zeigt es nach der Umschaltung in den PVA-Mode unverändert auf das gleiche Speichersegment.

Nach der Umschaltung in den PVA-Mode, spätestens jedoch vor der ersten Veränderung eines der Segmentregister, muß das Register GDTR auf eine gültige Globaldeskriptortabelle in einem RAM-Speicherbereich zeigen. Es bietet sich hier an, eine temporäre GDT im Speicher aufzubauen, über die in der Initialisierungsphase der Zugriff auf den vollen Adreßraum erfolgen kann. Diese temporäre GDT sollte nur die Einträge enthalten, die notwendig sind, um eine vorgefertigte Globaldeskriptortabelle, eine Interruptdeskriptortabelle, ein oder mehrere Task-State-Segmente und eventuelle Lokaldeskriptortabellen in den gewünschten RAM-Speicherbereich zu kopieren. Diese vorgefertigten Tabellen, die i.allg. mit speziellen Softwareentwicklungswerkzeugen (Systembuilder) erstellt werden, können entweder im EPROM oder durch eine Boot-Prozedur bereitgestellt werden.

14.3. Initialisierungsbeispiel im PVA-Mode

Im Anhang D ist ein Beispielprogramm für die Systeminitialisierung eines Softwaresystems, das im PVA-Mode arbeiten soll, abgebildet. In diesem Beispiel wurde besonderen Wert darauf gelegt, daß die Schutzmechanismen des Mikroprozessors 80286 möglichst weitgehend ausgenutzt werden. Deshalb erfolgt die Initialisierung fast ausschließlich im PVA-Mode.
Sicher muß nicht in jedem Fall eine Systeminitialisierung so aufwendig programmiert werden. Sie bietet jedoch die Gewähr, daß die Systemtabellen (Globaldeskriptortabelle, Interruptdeskriptortabelle, Task-State-Segment und Lokaldeskriptortabelle) vollständig eingerichtet sind und der Zugriff auf alle gewünschten Speichersegmente ohne Fehler erfolgen kann.

Im folgenden Abschnitt soll der Aufbau der Systemtabellen und der Initialisierungsablauf dieses Beispielprogramms kommentiert werden. Das Bild 14.1 gibt einen Überblick über den Algorithmus der Initialiserung.
Die Systemtabellen des Mikroprozessors 80286 (GDT, IDT, TSS, LDT) müssen

sich immer in einem RAM-Speicherbereich befinden, da sie vom Mikroprozessor 80286 selbst manipuliert werden (Accessbyte der Deskriptoren). In den meisten Softwaresystemen werden diese Tabellen während der Programmentwicklung mit Hilfe spezieller Entwicklungswerkzeuge (Systembuilder) konfiguriert. Die so vorgefertigten Tabellen müssen während der Initialisierung aus dem EPROM-Bereich des Rechners oder über eine Boot-Prozedur von einem externen Speichermedium in den Systemspeicherbereich übernommen werden.

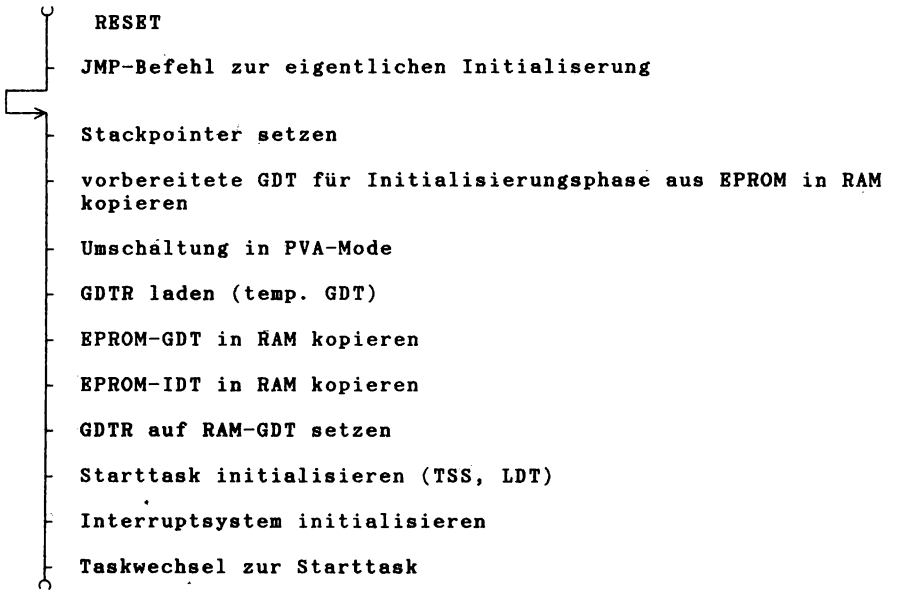


Bild 14.1. Algorithmus der Systeminitialisierung eines 80286-Softwaresystems im PVA-Mode (Beispiel)

Das Bild 14.2 enthält die im Beispielprogramm benutzten Systemtabellen und stellt die Zusammenhänge zwischen den einzelnen Tabellen dar.
Für die Initialisierungsphase ist eine temporäre GDT in einem RAM-Speicherbereich notwendig. Für diese temporäre GDT wurde ein EPROM-Vorbild erstellt, das zu einem möglichst frühen Zeitpunkt (im RA-Mode) aus dem EPROM in den RAM auf die physische Adresse 0 kopiert wird. Diese Tabelle im EPROM wird anschließend nicht mehr benutzt. Jeder weitere Verweis auf die EPROM-GDT bezieht sich auf die vorbereitete Globaldeskriptortabelle, die für das zu installierende Softwaresystem konfiguriert wurde. Diese vorbereitete EPROM-GDT erscheint - wie auch die EPROM-IDT, das EPROM-TSS und die EPROM-LDT der ersten Task - nicht im Listing des Beispielprogramms. Die Tabellen können auf beliebigen Speicheradressen im EPROM-Bereich angeordnet werden.

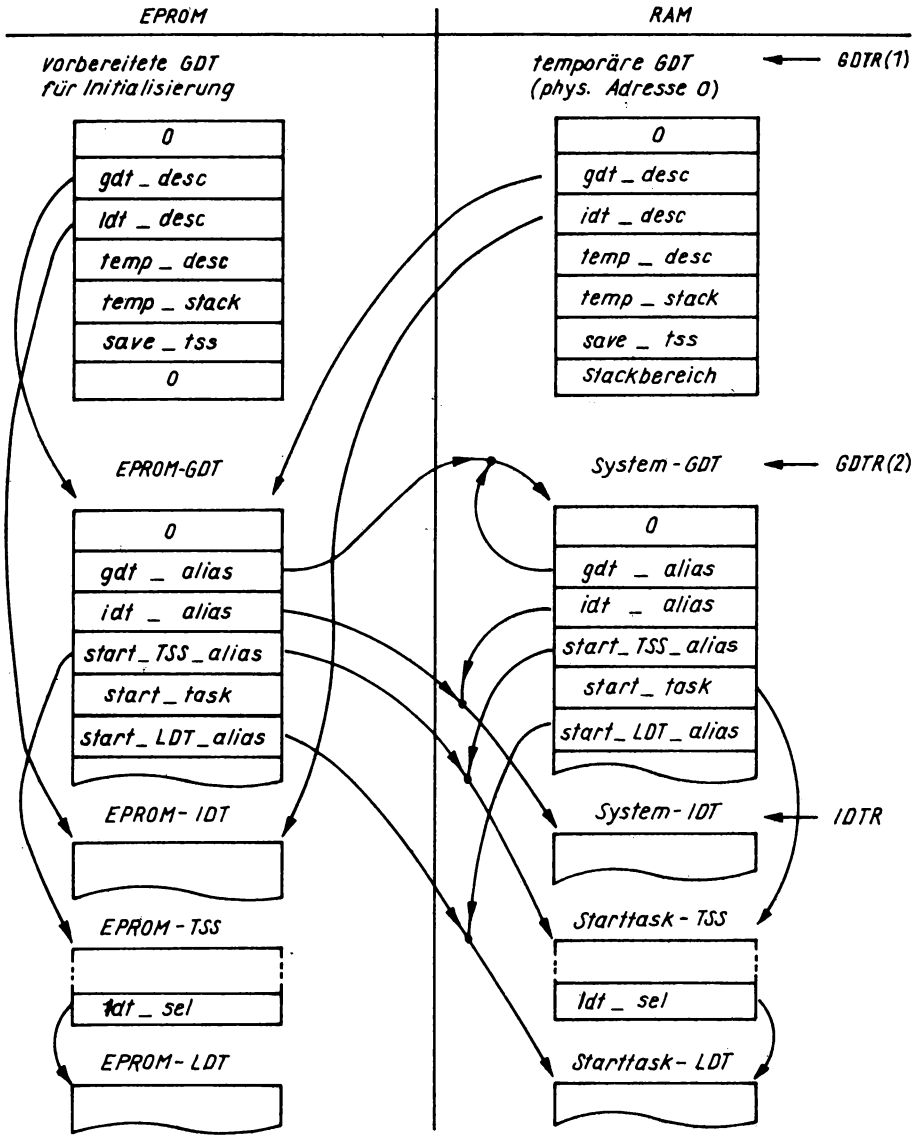


Bild 14.2. Systemtabellen für Initialisierungsbeispiel

Der temporären GDT, der EPROM-GDT wie auch der späteren System-GDT im RAM ist folgendes gemeinsam:

- der erste Deskriptor in einer Globaldeskriptortabelle ist ein Nulldeskriptor,

- der zweite Deskriptor nach dem Nullselektor beschreibt das Aliasdatensegment der Globaldeskriptortabelle,

- der dritte Deskriptor beschreibt das Alias-Datensegment der IDT.

Der Deskriptor des GDT-Alias-Datensegments in der temporären Globaldeskriptortabelle beschreibt das Datensegment, in dem die vorbereitete EPROM-GDT des Softwaresystems abgelegt ist. Dagegen beschreibt der GDT-Alias-Deskriptor im EPROM in der EPROM-GDT den Speicherbereich, in dem die GDT im zu installierenden Softwaresystem - also im RAM - zu einem späteren Zeitpunkt errichtet werden soll.
Somit kann auf die EPROM-GDT direkt über den Selektor

```
gdt_desc-initial_gdt
```

zugegriffen werden, während für den Zugriff auf das RAM-Datensegment, das die Globaldeskriptortabelle später aufnehmen soll, der entsprechende Deskriptor in der temporären GDT erst erstellt werden muß. Der Inhalt für diesen Deskriptoreintrag wird aus dem GDT-Alias-Deskriptor, der in der EPROM-GDT enthalten ist, kopiert (siehe Unterprogramm copy_EPROM_dt). Analog wird beim Kopieren der Interruptdeskriptortabelle (IDT) verfahren.

Nachdem die Systemtabellen GDT und IDT im RAM-Speicherbereich installiert sind, können die entsprechenden Register GDTR und IDTR neu geladen werden. Die Errichtung der ersten Task erfolgt erst, nachdem die System-GDT im RAM-Speicherbereich errichtet worden ist. In dieser Globaldeskriptortabelle müssen bereits Einträge für ein Task-State-Segment, ein Alias-Datensegment für dieses Task-State-Segment sowie - falls erforderlich - ein Alias-Datensegment für eine Lokaldeskriptortabelle vorhanden sein. Die vorbereiteten Segmente im EPROM - das Task-State-Segment sowie die Lokaldeskriptortabelle - werden in den RAM kopiert. Anschließend sind die entsprechenden Deskriptoreinträge für diese Segmente in der aktuellen Globaldeskriptortabelle (im RAM) zu aktualisieren.

Der Start der ersten Task erfolgt über einen Sprung zu dem im RAM befindlichen Task-State-Segment. Bei diesem ersten Taskwechsel wird der Kontext der "alten" Task im aktuellen Task-State-Segment (d.h., dem Speichersegment, das durch das Taskregister beschrieben wird) abgelegt. Im Beispielprogramm wird hier der Speicherbereich benutzt, in dem die temporäre GDT, die zu diesem Zeitpunkt nicht mehr benötigt wird, liegt.

15. Kompatibilität

Wie bereits in der Einleitung gesagt, reiht sich der Mikroprozessor 80286 in eine Entwicklungslinie hochintegrierter Mikroprozessoren ein. In diesem Abschnitt sollen die Randbedingungen zusammengefaßt werden, die zu beachten sind, um erarbeitete Software sowohl auf dem Vorgänger des 80286, dem 16-Bit-Mikroprozessor 8086, als auch auf dem 32-Bit-Mikroprozessor 80386 abarbeiten zu können.

15.1. Kompatibilität zum 16-Bit-Mikroprozessor 8086

Generell gilt, daß sich der Mikroprozessor 80286 in der Betriebsart REAL ADDRESS MODE so wie der Mikroprozessor 8086 verhält. Es sind jedoch die folgenden geringfügigen Unterschiede zu berücksichtigen:

1. Die reservierten Interruptvektoren 5, 6, 7, 8, 9, 13 und 16 (Abschn. 9.3. Reservierte Interruptvektoren) sollten in 8086-Programmen nicht benutzt werden, da sie für spezielle Exceptions des 80286 gedacht sind. Normalerweise treten in 8086-Programmen diese Interrupts nicht auf. Es ist jedoch zu empfehlen, in Anwenderprogrammen eine Interruptbehandlungsroutine einzubauen, die diese Exceptions als unzulässige Operationen auswertet.

2. Da sich die Anzahl der Takte, die für die Ausführung der einzelnen Befehle benötigt werden, bei beiden Mikroprozessoren unterscheidet, sollten Programmschleifen zur Erzeugung von Wartezeiten keine Anwendung finden. Zeiten sollten immer über geeignete Peripherieschaltkreise (Timer, Echtzeituhr) erzeugt werden.

3. Tritt beim Mikroprozessor 80286 eine Exception ein, zeigt der im Stack gerettete Wert der Register CS:IP immer auf den Beginn des Befehls, der die Exception verursachte. Der Mikroprozessor 8086 verhält sich im Fall einer Divisions-Exception anders. Hier zeigen die im Stack abgelegten Werte für CS:IP auf die Adresse des nächsten Befehls.

4. Wird der Mikroprozessor 80286 mit einem Numerikcoprozessor 80287 betrieben, muß die Behandlung von Fehlern im Zusammenhang mit dem Coprozessor immer über den reservierten Interrupt 16 erfolgen. Wurden in einem 8086/8087-System andere Vektoren benutzt, muß beim 80286 zusätzlich die dem Interrupt 16 zugeordnete Startadresse ebenfalls auf die Fehlerbehandlungsroutine zeigen.

5. Beim Mikroprozessor 80286 ist es möglich, ESC-Befehle mit Präfixen zu versehen. Tritt im Zusammenwirken mit dem Numerikcoprozessor 80287 eine Exception ein, zeigt der gerettete Wert für die Register CS:IP, der im vorgesehenen Rettungsbereich für die Coprozessorumgebung abgelegt wird, auf das erste führende Präfix vor dem ESC-Befehl. Dagegen zeigt der gerettete CS:IP-Wert in einem 8086/8087-System auf den ESC-Befehl selbst.

6. Soll ein Anwenderprogrammsystem auch auf dem Mikroprozessor 8086 lauffähig sein, dürfen selbstverständlich nur Befehle benutzt werden, die für diesen Prozessor zugelassen sind. Der Mikroprozessor 80286 erlaubt im RA-Mode gegenüber dem Mikroprozessor 8086 zusätzlich die Ausführung neuer bzw. erweiterter Befehle.

7. Nach einem RESET sind die Register CS:IP bei den Mikroprozessoren 8086 und 80286 unterschiedlich gesetzt:

   beim 8086 FFFF:0000
   beim 80286 F000:FFF0

   Dieser Unterschied kann korrigiert werden, indem auf der Adresse FFFF0H ein Befehl JMP FAR zur Initialisierungsroutine des Softwaresystems erfolgt.

8. Bei der Abarbeitung des Befehls PUSH SP wird beim Mikroprozessor 80286 der aktuelle Stackpointer im Stack abgelegt, während der 8086 den neuen Stackpointerwert (d.h., nach der Befehlsabarbeitung) abkellert.
   Um gleiche Wirkungen zu erzielen, sollte dieser Befehl durch folgende 3 Befehle ersetzt werden:

```
PUSH    BP
MOV     BP, SP
XCHG    BP, [BP]
```

9. Bei Rotations- und Verschiebebefehlen kann die Anzahl der Rotationen bzw. Verschiebeoperationen vorgegeben werden. Bei Mikroprozessor 80286 wird diese Anzahl zur Sicherung einer vertretbaren Interruptreaktionszeit eines Softwaresystems auf maximal 31 begrenzt (Maskierung der vorgegebenen Anzahl mit dem Wert 1FH).

10. Beim Mikroprozessor 80286 ist die Länge eines jeden Befehls auf 10 Byte begrenzt. Die einzige Möglichkeit, dieses Limit zu überschreiten, besteht darin, Präfixe vor einem Befehl mehrfach anzugeben. Bei Erkennung von Befehlen, die länger als 10 Byte sind, reagiert der 80286 mit einer Exception 6 (ungültiger Befehlskode). Für den Mikroprozessor 8086 existiert keine Beschränkung der Länge einzelner Befehle.

11. Das Präfix LOCK sollte in Verbindung mit dem korrespondierenden Ausgangssignal des Prozessors ausschließlich dazu benutzt werden, um Unterbrechungen während der Ausführung von Datentransferaktionen in einem busgekoppelten Multimastersystem zu verhindern.
    Der Mikroprozessor 80286 setzt das Signal /LOCK bei der Ausführung jedes XCHG-Befehls, auch dann, wenn das Präfix LOCK nicht angegeben ist. Dagegen setzt der 80286 das Signal /LOCK nicht während einer Befehlsleseoperation.

12. Der Einzelschrittinterrupt beim Mikroprozessor 80286 besitzt gegenüber dem 8086 eine höhere Priorität als jeder externe Interrupt. Damit wird verhindert, daß während der Abarbeitung von Programmen im Einzelschrittbetrieb externe Interrupts akzeptiert werden. Beim 8086 führte ein externer Interrupt während der Testung einer Applikation im Einzelschrittbetrieb zur Abarbeitung der Behandlungsroutine des externen Interrupts im Einzelschrittbetrieb.

13. Wird bei Divisionen mit Hilfe des Befehls IDIV das Ergebnis zu groß (Ergebnis kann nicht im entsprechenden Integerformat im Zieloperanden aufgenommen werden), kann der 80286 als Quotient die betragsmäßig größte, negative Zahl (80H bzw. 8000H) erzeugen, während der 8086 Exception 0 generiert.

14. Nach Anerkennung eines nichtmaskierbaren Interrupts (NMI) wird der NMI-Eingang ebenso wie der Interrupt bei Bereichsüberschreitung im Zusammenwirken mit dem Numerikcoprozessor so lange maskiert, bis der erste IRET-Befehl ausgeführt wurde.

15. Das Fehlersignal des Numerikcoprozessors 80287 wird im Zusammenspiel mit dem 80286 nicht über einen Interruptcontroller geführt. Es sind deshalb keine Befehle zur Programmierung des Interruptcontrollerbausteins zu verwenden.

16. Anwenderprogramme für den Mikroprozessor 8086 können einen Überlauf des Adreßbereichs gewollt ausnutzen (z.B. FFF0:0500 entspricht dann 0000:0400). Beim 80286 muß hier eine externe Hardware dafür sorgen, daß die höchstwertigen vier Bitpositionen der 24-Bit-Adresse im RA-Mode auf Null gesetzt werden.

17. Die Ein-/Ausgabe-Adressen 00F8H bis 00FFH sind reserviert.

15.2. Kompatibilität zum 32-Bit-Mikroprozessor 80386

Generell gilt, daß bei Einhaltung aller Beschränkungen und Empfehlungen, die in den bisherigen Abschnitten zur Programmierung des Mikroprozessors 80286 angeführt wurden, sämtliche 80286-Programme auf dem Mikroprozessor 80386 lauffähig sind. Folgende Randbedingungen sind jedoch zur Sicherung der Kompatibilität 80286/80386 zu beachten:

1. Die Programmteile zur Initialisierung des PVA-Modes nach einem RESET sind vom Anwendersystem zu isolieren, da vor der Ausführung von 80286-Programmteilen verschiedene Operationsparameter des 80386 zu setzen sind.
   Ein Beispiel für ein isoliertes Initialisierungsprogramm ist das Programmfragment, das im Abschnitt 14. Systeminitialisierung und im Anhang D vorgestellt wird. Die notwendigen Befehle können hier problemlos eingefügt werden.

2. Kompatibilitätsprobleme können auftreten, wenn in Programmen Modifikationen des eigenen Programmkodes vorgenommen werden sollen. Der Mikroprozessor 80386 verfügt über die Möglichkeit der Verwaltung

einer größeren Befehlskodewarteschlange (instruction cache). Somit kann der Fall eintreten, daß der Befehl, der modifiziert werden soll, sich bereits in der Warteschlange befindet und der Prozessor die Modifikation nicht mehr erfaßt. Können Befehlskodemodifikationen nicht umgangen werden (z.B. bei Debuggerprogrammen), muß im Zusammenhang mit der Modifikation die Befehlswarteschlange gelöscht werden (z.B. durch Einfügen eines Sprungbefehls).

3. Es ist zu beachten, daß sich der Mikroprozessor 80386 bei der Abarbeitung von Programmen, bei denen es zu einer Überschreitung des physischen Adreßraums des 80286 von 16 MByte kommt, aufgrund seines wesentlich größeren Adreßraums anders verhält als der 80286. Während der 80386 in diesem Fall den Adreßbereich ab 1000000H (17. MByte) anspricht, erfolgt beim 80286 ein Umschlag auf die Adressen ab 0H.

4. Die reservierten Wörter aller Segment- oder Systemdeskriptoren in 80286-Programmen müssen den Wert 0 beinhalten. Andere Inhalte werden vom 80386 ausgewertet und können zu Fehlinterpretationen führen.

5. Unbenutzte Deskriptoren in den Systemtabellen GDT, IDT und LDT sind im Accessbyte durch die Werte 80H oder 0H zu kennzeichnen. Andere Werte können vom 80386 als gültige Deskriptortypen erkannt werden.

6. Der Mikroprozessor 80386 benutzt ein anderes Protokoll zur Realisierung der BUSLOCK-Funktion. Das LOCK-Präfix sollte auch hier nur benutzt werden, wenn Datentransferoperationen in Multimastersystemen vor Unterbrechungen zu schützen sind.
In einem 80386-System kann das LOCK-Präfix nur in Verbindung mit den Befehlen INC, DEC, NOT, NEG, ADD, ADC, SUB, SBB, AND, OR, XOR benutzt werden (hier sind nur die Befehle aufgeführt, die der Mikroprozessor 80286 ebenfalls ausführen kann). Wird das LOCK-Präfix in Verbindung mit anderen Befehlen ausgeführt, resultiert die Exception 6 (ungültiger Befehlskode).

7. Werden 80286-Programme in einem 80386-System in einer Betriebssystemumgebung abgearbeitet, in der das sogenannte "Paging" erlaubt ist, kann unter bestimmten Konstellationen die Exception 14 (page fault) eintreten. Es sollte deshalb für diese Exception eine Interruptbehandlungsroutine vorgesehen werden.

Befehl	Erläuterung	Operanden	Format	Takte	Flags O S Z A P C	Bemerkungen
Präfixe						
REP MOVS	MOVS; CX:=CX-1 (bis CX=0)	M,M	11110011 | 1010010w	5+4n	- - - - - -	
REP LODS	LODS; CX:=CX-1 (bis CX=0)	M	11110011 | 1010110w	5+4n	- - - - - -	
REP STOS	STOS; CX:=CX-1 (bis CX=0)	M	11110011 | 1010101w	4+3n	- - - - - -	
REP INS	INS; CX:=CX-1 (bis CX=0)	M	11110011 | 0110110w	5+4n	- - - - - -	
REP OUTS	OUTS; CX:=CX-1 (bis CX=0)	M	11110011 | 0110111w	5+4n	- - - - - -	
REPE CMPS	CMPS; CX:=CX-1 (bis CX=0 o. ZF=0)	M,M	11110011 | 1010011w	5+9n	M M M M M M	entspr. REPZ
REPE SCAS	SCAS; CX:=CX-1 (bis CX=0 o. ZF=0)	M	11110011 | 1010111w	5+8n	M M M M M M	entspr. REPZ
REPNE CMPS	CMPS; CX:=CX-1 (bis CX=0 o. ZF=1)	M,M	11110010 | 1010011w	5+9n	M M M M M M	entspr. REPNZ
REPNE SCAS	SCAS; CX:=CX-1 (bis CX=0 o. ZF=1)	M	11110010 | 1010111w	5+8n	M M M M M M	entspr. REPNZ
LOCK	BUSLOCK aktiv für nächsten Befehl		11110000	0	- - - - - -	
-	Segment-Override-Präfix		001sr110	0	- - - - - -	

Befehl	Erläuterung	Operanden	Format	Takte	Flags O S Z A P C	Bemerkungen
JCXZ cb	CX=0: IP:=IP+cb	cb	11100011 dblo	8+a,noj=4	- - - - - -	
Prozessorsteuerbefehle						
STC	CF:=1	-	11111001	2	- - - - - 1	
CLC	CF:=0	-	11111000	2	- - - - - 0	
CMC	CF:=/CF	-	11110101	2	- - - - - M	
STD	DF:=1	-	11111101	2	- - - - - -	
CLD	DF:=0	-	11111100	2	- - - - - -	
STI	IF:=1	-	11111011	2	- - - - - -	Vor.: CPL <= IOPL
CLI	IF:=0	-	11111010	3	- - - - - -	Vor.: CPL <= IOPL
HLT	Keine Befehlsabarbeitung	-	11110100	2	- - - - - -	Vor.: CPL = 0
WAIT	Wait, bis /BUSY inaktiv	-	10011011	3	- - - - - -	Vor.: CPL = 0
CLTS	NT:=0	-	00001111 00000110	2	- - - - - -	Vor.: CPL = 0
ESC	Coprozessorbefehle		11011TTT moLLLr/m	9-20	- - - - - -	
Steuerbefehle für den Schutzmechanismus						
LGDT m	GDTR.limit:=m; GDTR.base:=m+2	M	00001111 00000001 mo010r/m	11	- - - - - -	Vor.: CPL = 0
SGDT m	m:=GDTR.limit; m+2:=GDTR.base	M	00001111 00000001 mo000r/m	11	- - - - - -	
LIDT m	IDTR.limit:=m; IDTR.base:=m+2	M	00001111 00000001 mo011r/m	12	- - - - - -	Vor.: CPL = 0
SIDT m	m:=IDTR.limit; m+2:=IDTR.base	M	00001111 00000001 mo001r/m	12	- - - - - -	
LLDT ew	LDTR:=GDT(ew)	ew	00001111 00000000 mo010r/m	17,m=19	- - - - - -	Vor.: CPL = 0
SLDT ew	ew:=LDTR (nur Selektor)	ew	00001111 00000000 mo000r/m	2,m=3	- - - - - -	
LTR ew	TR:=ew (Selektor)	ew	00001111 00000000 mo011r/m	17,m=19	- - - - - -	Vor.: CPL = 0
STR ew	ew:=TR	ew	00001111 00000000 mo001r/m	2,m=3	- - - - - -	
LMSW ew	MSW:=ew	ew	00001111 00000001 mo110r/m	3,m=6	- - - - - -	Vor.: CPL = 0
SMSW ew	ew:=MSW	ew	00001111 00000001 mo100r/m	2,m=3	- - - - - -	

Befehl	Erläuterung	Operanden	Format	Takte	Flags O S Z A P C	Bemerkungen
LAR rw,ew	hi(rw):=AR-Byte(Selektor ew)	rw,ew	00001111 | 00000010 | moregr/m	14,m=16	- - M - - -	
LSL rw,ew	rw:=Segm.limit (Selektor ew)	rw,ew	00001111 | 00000011 | moregr/m	14,m=16	- - M - - -	
ARPL ew,rw	RPL(ew)<RPL(rw): ZF=1 && RPL(ew):=RPL(rw)	ew,rw	01100011 | moregr/m	10,m=11	- - M - - -	
VERR ew	Segment ew lesbar: ZF:=1	ew	00001111 | 00000000 | mo100r/m	14,m=16	- - M - - -	
VERW ew	Segm. ew beschreibbar: ZF:=1	ew	00001111 | 00000000 | mo101r/m	14,m=16	- - M - - -	
Komplexe Befehle						
ENTER dw,0	PUSH BP; BP:=SP; SP:=SP-dw	dw,0	11001000 | dblo | dbhi | 00000000	11	- - - - - -	
ENTER dw,1	Stackrahmen anlegen	dw,1	11001000 | dblo | dbhi | 00000001	15	- - - - - -	
ENTER dw,dblev	Stackrahmen anlegen	dw,db	11001000 | dblo | dbhi | dblev	12+4dblev	- - - - - -	dblev <= 31
LEAVE	SP:=BP; POP BP	-	11001001	5	- - - - - -	
BOUND rw,md	rw<m :: rw>m+2: INT 5	rw,md	01100010 | moregr/m	noj=13	- - - - - -	

Befehl	Erläuterung	Operanden	Format	Takte	Flags O S Z A P C	Bemerkungen
CALL FAR dst	(dst: indirekte Adr.)	ed	11111111 | mo011r/m	16,pm=29+a	- - - - - -	
	(dst: CALL-Gate indirekt)			pm=44+a	- - - - - -	DPL = CPL
				pm=83+a	- - - - - -	DPL < CPL
				pm=90+4x+a	- - - - - -	DPL < CPL, x Param.
	(dst: TSS indirekt)			pm=180+a	M M M M M M	Taskwechsel
	(dst: Task-Gate indirekt)			pm=185+a	M M M M M M	Taskwechsel
RET NEAR	IP:=[SP]; SP:=SP+2	-	11000011	11+a	- - - - - -	
RET NEAR dw	IP:=[SP]; SP:=SP+2+2dw	dw	11000010 | dblo | dbhi	11+a	- - - - - -	
RET FAR	CS:IP=[SP]; SP=SP+4	-	11001011	15,pm=25+a	- - - - - -	DPL = CPL
				pm=55+a	- - - - - -	DPL > CPL
RET FAR dw	CS:IP=[SP]; SP=SP+4+2dw	dw	11001010 | dblo | dbhi	15,pm=25+a	- - - - - -	DPL = CPL
				pm=55+a	- - - - - -	DPL > CPL
INT 3	Interrupt 3 ; TF=0; IF=0	-	11001100	ra=23+a	- - - - - -	Debuggertrap
	TF=0; IF=0 falls			pm=40+a	- - - - - -	DPL = CPL
	Interrupt-Gate			pm=78+a	- - - - - -	DPL < CPL
	(Task-Gate)			pm=167+a	M M M M M M	Taskwechsel
INT db	Interrupt db ; TF=0; IF=0	db	11001101 | dblo	ra=23+a	- - - - - -	
	TF=0; IF=0 falls			pm=40+a	- - - - - -	DPL = CPL
	Interrupt-Gate			pm=78+a	- - - - - -	DPL < CPL
	(Task-Gate)			pm=167+a	M M M M M M	Taskwechsel
INTO	OF=1: Interrupt 4	-	11001110	24,noj=3+a	- - - - - -	
IRET	Rückkehr vom Interrupt	-	11001111	17,pm=31+a	M M M M M M	DPL = CPL
				pm=55+a	M M M M M M	DPL > CPL
				pm=169+a	M M M M M M	andere Task (NT=1)
JMP SHORT cb	IP:=IP+cb	cb	11101011 | dblo	7+a	- - - - - -	innerhalb Segment
JMP NEAR dst	IP:=IP+dst (Direktadr.)	cw	11101001 | dblo | dbhi	7+a	- - - - - -	innerhalb Segment
	IP:=[dst] (indirekte Adr.)	ew	11111111 | mo100r/m	7,m=11+a	- - - - - -	innerhalb Segment

Befehl	Erläuterung	Operanden	Format	Takte	Flags O S Z A P C	Bemerkungen
JMP FAR dst	CS:IP:=dst (Direktadr.)	cd	11101010 | Segmentoffset Segmentselektor	11,pm=23+a	- - - - - -	anderes Segment
	(dst:CALL-Gate)			pm=38+a	- - - - - -	DPL = CPL
	(dst:TSS)			pm=175+a	M M M M M M	
	(dst:Task-Gate)			pm=180+a	M M M M M M	
	CS:IP:=[dst] (indir. Adr.)	ed	11111111 | mo101r/m	15,pm=26+a	- - - - - -	
	(dst:CALL-Gate, indirekt)			pm=41+a	- - - - - -	DPL = CPL
	(dst:TSS, indirekt)			pm=178+a	M M M M M M	
	(dst:Task-Gate, indirekt)			pm=183+a	M M M M M M	
JE/JZ cb	ZF=1: IP:=IP+cb	cb	01110100 | dblo	7+a,noj=3	- - - - - -	
JL/JNGE cb	SF!=OF: IP:=IP+cb	cb	01111100 | dblo	7+a,noj=3	- - - - - -	
JLE/JNG cb	(ZF=1)||(SF!=OF): IP:=IP+cb	cb	01111110 | dblo	7+a,noj=3	- - - - - -	
JB/JNAE cb	CF=1: IP:=IP+cb	cb	01110010 | dblo	7+a,noj=3	- - - - - -	
JBE/JNA cb	(CF=1)||(ZF=1): IP:=IP+cb	cb	01110110 | dblo	7+a,noj=3	- - - - - -	
JP/JPE cb	PF=1: IP:=IP+cb	cb	01111010 | dblo	7+a,noj=3	- - - - - -	
JO cb	OF=1: IP:=IP+cb	cb	01110000 | dblo	7+a,noj=3	- - - - - -	
JS cb	SF=1: IP:=IP+cb	cb	01111000 | dblo	7+a,noj=3	- - - - - -	
JNE/JNZ cb	ZF=0: IP:=IP+cb	cb	01110101 | dblo	7+a,noj=3	- - - - - -	
JNL/JGE cb	SF=OF: IP:=IP+cb	cb	01111101 | dblo	7+a,noj=3	- - - - - -	
JNLE/JG cb	(ZF=0)&&(SF=OF): IP:=IP+cb	cb	01111111 | dblo	7+a,noj=3	- - - - - -	
JNB/JAE cb	CF=0: IP:=IP+cb	cb	01110011 | dblo	7+a,noj=3	- - - - - -	
JNBE/JA cb	(CF=0)&&(ZF=0): IP:=IP+cb	cb	01110111 | dblo	7+a,noj=3	- - - - - -	
JNP/JPO cb	PF=0: IP:=IP+cb	cb	01111011 | dblo	7+a,noj=3	- - - - - -	
JNO cb	OF=0: IP:=IP+cb	cb	01110001 | dblo	7+a,noj=3	- - - - - -	
JNS cb	SF=0: IP:=IP+cb	cb	01111001 | dblo	7+a,noj=3	- - - - - -	
LOOP cb	CX!=0: IP:=IP+cb	cb	11100010 | dblo	8+a,noj=4	- - - - - -	
LOOPZ/E cb	(CX!=0)&&(ZF=1): IP:=IP+cb	cb	11100001 | dblo	8+a,noj=4	- - -	
LOOPNZ/NE cb	(CX!=0)&&(ZF=0): IP:=IP+cb	cb	11100000 | dblo	8+a,noj=4	- - - - - -	

Befehl	Erläuterung	Operanden	Format	Takte	Flags O S Z A P C	Bemerkungen
OR dst,	dst:=dst OR src	EA,R	000010dw | moregr/m	2,m=7	O M M U M O	
		EA,IM	1000000w | mo001r/m | dblo | dbhi	3,m=7	O M M U M O	
		A,IM	0000110w | dblo | dbhi	3	O M M U M O	
XOR dst,src	dst:=dst XOR src	EA,R	001100dw | moregr/m	2,m=7	O M M U M O	
		EA,IM	1000000w | mo110r/m | dblo | dbhi	3,m=7	O M M U M O	
		A,IM	0011010w | dblo | dbhi	3	O M M U M O	
NOT dst	dst:=NOT dst	EA	1111011w | mo010r/m	2,m=7	- - - - - -	
Rotations-/Verschiebebefehle						
ROL dst,src		EA,1	1101000w | mo000r/m	2,m=7	M - - - - M	
		EA,CL	1101001w | mo000r/m	5+n,m=8+n	M*- - - - M	
		EA,db	1100000w | mo000r/m | dblo	5+n,m=8+n	M*- - - - M	
ROR dst,src		EA,1	1101000w | mo001r/m	2,m=7	M - - - - M	
		EA,CL	1101001w | mo001r/m	5+n,m=8+n	M*- - - - M	
		EA,db	1100000w | mo001r/m | dblo	5+n,m=8+n	M*- - - - M	
RCL dst,src		EA,1	1101000w | mo010r/m	2,m=7	M - - - - M	
		EA,CL	1101001w | mo010r/m	5+n,m=8+n	M*- - - - M	
		EA,db	1100000w | mo010r/m | dblo	5+n,m=8+n	M*- - - - M	
RCR dst,src		EA,1	1101000w | mo011r/m	2,m=7	M - - - - M	
		EA,CL	1101001w | mo011r/m	5+n,m=8+n	M*- - - - M	
		EA,db	1100000w | mo011r/m | dblo	5+n,m=8+n	M*- - - - M	
SAL dst,src		EA,1	1101000w | mo100r/m	2,m=7	M M M U M M	entspricht SHL
		EA,CL	1101001w | mo100r/m	5+n,m=8+n	M*M M U M M	
		EA,db	1100000w | mo100r/m | dblo	5+n,m=8+n	M*M M U M M	
SHR dst,src		EA,1	1101000w | mo101r/m	2,m=7	M M M U M M	
		EA,CL	1101001w | mo101r/m	5+n,m=8+n	M*M M U M M	
		EA,db	1100000w | mo101r/m | dblo	5+n,m=8+n	M*M M U M M	

Befehl	Erläuterung	Operanden	Format	Takte	Flags O S Z A P C	Bemerkungen
SAR dst,		EA,1	1101000w | mo111r/m	2,m=7	O M M U M M	
		EA,CL	1101001w | mo111r/m	5+n,m=8+n	M*M M U M M	
		EA,db	1100000w | mo111r/m | dblo	5+n,m=8+n	M*M M U M M	
Zeichenkettenmanipulationsbefehle						
MOVS	ES:[DI]:=[SI]; SI++, DI++	M,M	1010010w	5	- - - - - -	DF | w=0 | w=1 0 | R:=R+1 | R:=R+2 1 | R:=R-1 | R:=R-2
CMPS	ES:[DI]-[SI] ? SI++, DI++	M,M	1010011w	8	M M M M M M	
SCAS	A-ES:[DI] ; DI++	M	1010111w	7	M M M M M M	
LODS	A:=[SI] ; SI++	M	1010110w	5	- - - - - -	
STOS	ES:[DI]:=A ; DI++	M	1010101w	3	- - - - - -	
INS	ES:[DI]:=[DX]; DI++	M	0110110w	5	- - - - - -	Vor.: CPL <= IOPL
OUTS	[DX]:=[SI]; SI++	M	0110111w	5	- - - - - -	Vor.: CPL <= IOPL
Programmsteuerbefehle						
CALL NEAR dst	UP-Aufruf (dst: Direktadr.)	cw	11101000 | dblo | dbhi	7+a	- - - - - -	innerhalb Segment
	(dst: indirekt Adr.)	ew	11111111 | mo010r/m	7,m=11+a	- - - - - -	innerhalb Segment
CALL FAR dst	UP-Aufruf (dst: Direktadr.)	cd	10011010 | Segmentoffset Segmentselektor	13,pm=26+a	- - - - - -	anderes Segment
	(dst: CALL-Gate)			pm=41+a	- - - - - -	DPL = CPL
				pm=82+a	- - - - - -	DPL < CPL
				pm=86+4x+a	- - - - - -	DPL < CPL, x Par.
	(dst: TSS)			pm=177+a	M M M M M M	Taskwechsel
	(dst: Task-Gate)			pm=182+a	M M M M M M	Taskwechsel

Befehl	Erläuterung	Operanden	Format	Takte	Flags O S Z A P C	Bemerkungen
NEG dst	dst:=-dst	EA	1111011w mo011r/m	2,m=7	M M M M M M	
AAA	ASCII-Korrektur nach ADD	-	00110111	3	U U U M U M	BCD ungepackt
DAA	Dezimalkorrektur nach ADD	-	00100111	3	U M M M M M	BCD gepackt
AAS	ASCII-Korrektur nach SUB	-	00111111	3	U U U M U M	BCD ungepackt
DAS	Dezimalkorrektur nach SUB	-	00101111	3	U M M M M M	BCD gepackt
MUL src	AX:=AL*eb	eb	11110110 mo100r/m	13,m=16	M U U U U M	vorzeichenlos
	DX:AX:=AX*ew	ew	11110111 mo100r/m	21,m=24	M U U U U M	
IMUL src	AX:=AL*eb	eb	11110110 mo101r/m	13,m=16	M U U U U M	vorzeichen-
	DX:AX:=AX*ew	ew	11110111 mo101r/m	21,m=24	M U U U U M	behaftet
IMUL rw,ew,src	rw:=ew*src	rw,ew,IM	011010s1 moregr/m dblo dbhi	21,m=24	M U U U U M	
DIV src	AL:=AX/eb ; AH=Rest	eb	11110110 mo110r/m	14,m=17	U U U U U U	vorzeichenlos
	AX:=DX:AX/ew ; DX=Rest	ew	11110111 mo110r/m	22,m=25	U U U U U U	
IDIV src	AL:=AX/eb ; AH=Rest	eb	11110110 mo111r/m	17,m=20	U U U U U U	vorzeichen-
	AX:=DX:AX/ew ; DX=Rest	ew	11110111 mo111r/m	25,m=28	U U U U U U	behaftet
AAM	ASCII-Korrektur nach MUL	-	11010100 00001010	16	U M M U M U	BCD ungepackt
AAD	ASCII-Korrektur vor DIV	-	11010101 00001010	14	U M M U M U	BCD ungepackt
CBW	AX:=AL (sign extension)	-	10011000	2	- - - - - -	
CWD	DX:AX:=AX (sign extension)	-	10011001	2	- - - - - -	
Logikbefehle						
AND dst,src	dst:=dst AND src	EA,R	001000dw moregr/m	2,m=7	0 M M U M 0	
		EA,IM	1000000w mo100r/m dblo dbhi	3,m=7	0 M M U M 0	
		A,IM	0010010w dblo dbhi	3	0 M M U M 0	
TEST dst,src	dst AND src ?	EA,R	1000010w moregr/m	2,m=6	0 M M U M 0	
	(nur Flagbeeinflussung)	EA,IM	1111011w mo000r/m dblo dbhi	3,m=6	0 M M U M 0	
		A,IM	1010100w dblo dbhi	3	0 M M U M 0	

Befehl	Erläuterung	Operanden	Format	Takte	Flags O S Z A P C	Bemerkungen
LEA rw,m	rw:=OFFSET (m)	rw,m	10001101 | moregr/m | adrlo | adrhi	3	- - - - - -	
LDS rw,ed	DS:rw:=ed	rw,ed	11000101 | moregr/m | adrlo | adrhi	7,pm=21	- - - - - -	mo != 11
LES dst,ed	ES:rw:=ed	rw,ed	11000100 | moregr/m | adrlo | adrhi	7,pm=21	- - - - - -	mo != 11
LAHF	AH:=FLAGS	-	10011111	2	- - - - - -	
SAHF	FLAGS:=AH	-	10011110	2	- M M M M M	
Arithmetikbefehle						
ADD dst,src	dst:=dst+src	EA,R	000000dw | moregr/m	2,m=7	M M M M M M	
		EA,IM	100000sw | mo000r/m | dblo | dbhi	3,m=7	M M M M M M	
		A,IM	0000010w | dblo | dbhi	3	M M M M M M	
ADC dst,src	dst:=dst+src+CF	EA,R	000100dw | moregr/m	2,m=7	M M M M M M	
		EA,IM	100000sw | mo010r/m | dblo | dbhi	3,m=7	M M M M M M	
		A,IM	0001010w | dblo | dbhi	3	M M M M M M	
INC dst	dst:=dst+1	EA	1111111w | mo000r/m	2,m=7	M M M M M -	
		rw	01000reg	2	M M M M M -	
SUB dst,src	dst:=dst-src	EA,R	001010dw | moregr/m	2,m=7	M M M M M M	
		EA,IM	100000sw | mo101r/m | dblo | dbhi	3,m=7	M M M M M M	
		A,IM	0010110w | dblo | dbhi	3	M M M M M M	
SBB dst,src	dst:=dst-src-CF	EA,R	000110dw | moregr/m	2,m=7	M M M M M M	
		EA,IM	100000sw | mo011r/m | dblo | dbhi	3,m=7	M M M M M M	
		A,IM	0001110w | dblo | dbhi	3	M M M M M M	
DEC dst	dst:=dst-1	EA	1111111w | mo001r/m	2,m=7	M M M M M -	
		rw	01001reg	2	M M M M M -	
CMP dst,src	dst-src ?	R,EA	0011101w | moregr/m	2,m=6	M M M M M M	
	(nur Flagbeeinflussung)	EA,R	0011100w | moregr/m	2,m=7	M M M M M M	
		EA,IM	100000sw | mo111r/m | dblo | dbhi	3,m=6	M M M M M M	
		A,IM	0011110w | dblo | dbhi	3	M M M M M M	

Befehl	Erläuterung	Operanden	Format	Takte	Flags O S Z A P C	Bemerkungen
Datentransferbefehle						
MOV dst,src	dst:=src	EA,R	1000100w | moregr/m	2,m=3	- - - - - -	
		R,EA	1000101w | moregr/m	2,m=5	- - - - - -	
		EA,IM	1100011w | mo000r/m | dblo | dbhi	2,m=3	- - - - - -	
		R,IM	1011wreg | dblo | dbhi	2	- - - - - -	
		A,X	1010000w | adrlo | adrhi	5	- - - - - -	
		X,A	1010001w | adrlo | adrhi	3	- - - - - -	
		sr,EA	10001110 | mo0srr/m	ra:2,m=5 pm:17,m=19	- - - - - -	sr != 01 (CS)
		EA,sr	10001100 | mo0srr/m	2,m=3	- - - - - -	
PUSH src	SP:=SP-2; [SS:SP]:=src	mw	11111111 | mo110r/m | adrlow | adrhi	5	- - - - - -	
		R	01010reg	3	- - - - - -	
		sr	000sr110	3	- - - - - -	
		IM	011010s0 | dblo | dbhi	3	- - - - - -	
PUSHA	PUSH all	-	01100000	17	- - - - - -	
PUSHF	SP:=SP-2; [SS:SP]:=FLAGS	-	10011100	3	- - - - - -	
POP dst	dst:=[SS:SP]; SP:=SP+2	mw	10001111 | mo000r/m | adrlow | adrhi	5	- - - - - -	
		R	01011reg	5	- - - - - -	
		sr	000sr111	5,pm=20	- - - - - -	sr != 01 (CS)
POPA	POP all	-	01100001	19	- - - - - -	
POPF	FLAGS:=[SS:SP]; SP:=SP+2	-	10011101	5	M M M M M M	
XCHG dst,src	Austausch dst-src	EA,R	1000011w | moregr/m	3,m=5	- - - - - -	
		A,R	10010reg	3	- - - - - -	
IN port	A:=[port]	A,IM	1110010w | port	5	- - - - - -	Vor.: CPL <= IOPL
	A:=[DX]	-	1110110w	5	- - - - - -	
OUT port	[port]:=A	IM,A	1110011w | port	3	- - - - - -	Vor.: CPL <= IOPL
	[DX]:=A	-	1110111w	3	- - - - - -	

Operanden

A	Akkumulator (AL oder AX)	dw	Direktwert (16 Bit)
EA	effektive Adresse (Byte oder Wort, R oder M)	eb	effektive Adresse (8 Bit, R oder M)
IM	Direktwert (Byte oder Wort)	ed	speicherbezogener Pointer (32 Bit)
M	allg. Speicheroperand (alle Adressierungsarten)	ew	effektive Adresse (16 Bit, R oder M)
R	Register (Byte oder Wort)	mb	Speichervariable (8 Bit, alle Adressierungsarten)
X	allg. Speicheroperand (ohne Index- oder Basisadr.)	mw	Speichervariable (16 Bit, alle Adressierungsarten)
dst	Ziel (destination)	port	Ein-/Ausgabe-Adresse
src	Quelle (source)	rb	Byteregister
cb	Distanzwert (8 Bit, Programmkode)	rw	Wortregister
cd	Programmkodeadresse (32 Bit)	sr	Segmentregister
cw	Distanzwert (16 Bit, Programmkode)	xb	Speichervariable (8 Bit, ohne Index- oder Basisadr.)
db	Direktwert (8 Bit)	xw	Speichervariable (16 Bit, ohne Index- oder Basisadr.)

Format

d	Datenrichtung zum Register (d=1)
	Datenrichtung vom Register (d=0)
w	Wortoperation (w=1)
	Byteoperation (w=0)
s	16-Bit-Direktwert (s=0)
	Vorzeichenexpansion (s=1)
reg	Registerkodierung
sr	Segmentregisterkodierung
mo	mod-Feld im ModRM-Byte
r/m	r/m-Feld im ModRM-Byte
adrlo	Low-Byte der eff. Adresse
adrhi	High-Byte der eff. Adresse
dblo	Low-Datenbyte
dbhi	High-Datenbyte (entfällt bei 8-Bit-Direktwerten)

Flags

O	Overflow
S	Sign
Z	Zero
A	Auxiliary Carry
P	Parity
C	Carry

Flagbeeinflussung

M	modifiziert
U	undefiniert
	unverändert
0,1	gesetzter Wert
M*	siehe Befehlsbeschreibung

Logik

!=	ungleich
|	ODER-Verknüpfung
&&	UND-Verknüpfung
++	Pointerbeeinflussung lt. Tabelle

Takte

m=	Speicheroperand
noj=	kein Sprung (no jump)
pm=	PROTECTED VIRTUAL ADDRESS MODE (PVA-Mode)
ra=	REAL ADDRESS MODE (RA-Mode)
+a	Add. je 1 Takt pro Byte d. nächsten Befehls
+n	Add. je 1 Takt pro Verschiebung/Rotation

Für alle indizierten Speicherzugriffe, die sowohl Basisregister, Indexregister als auch Distanzwert benutzen, ist 1 Takt zu addieren!

Anhang B. Übersicht ModRM-Byte-Kombinationen

rb =	AL	CL	DL	BL	AH	CH	DH	BH	
rw =	AX	CX	DX	BX	SP	BP	SI	DI	
reg =	0	1	2	3	4	5	6	7	
	ModRM-Werte								Effektive Adresse
mod=00	00	08	10	18	20	28	30	38	[BX + SI]
	01	09	11	19	21	29	31	39	[BX + DI]
	02	0a	12	1a	22	2a	32	3a	[BP + SI]
	03	0b	13	1b	23	2b	33	3b	[BP + DI]
	04	0c	14	1c	24	2c	34	3c	[SI]
	05	0d	15	1d	25	2d	35	3d	[DI]
	06	0e	16	1e	26	2e	36	3e	D16
	07	0f	17	1f	27	2f	37	3f	[BX]
mod=01	40	48	50	58	60	68	70	78	[BX + SI] + D8 (1)
	41	49	51	59	61	69	71	79	[BX + DI] + D8
	42	4a	52	5a	62	6a	72	7a	[BP + SI] + D8
	43	4b	53	5b	63	6b	73	7b	[BP + DI] + D8
	44	4c	54	5c	64	6c	74	7c	[SI] + D8
	45	4d	55	5d	65	6d	75	7d	[DI] + D8
	46	4e	56	5e	66	6e	76	7e	[BP] + D8 (2)
	47	4f	57	5f	67	6f	77	7f	[BX] + D8
mod=10	80	88	90	98	a0	a8	b0	b8	[BX + SI] + D16 (3)
	81	89	91	99	a1	a9	b1	b9	[BX + DI] + D16
	82	8a	92	9a	a2	aa	b2	ba	[BP + SI] + D16
	83	8b	93	9b	a3	ab	b3	bb	[BP + DI] + D16
	84	8c	94	9c	a4	ac	b4	bc	[SI] + D16
	85	8d	95	9d	a5	ad	b5	bd	[DI] + D16
	86	8e	96	9e	a6	ae	b6	be	[BP] + D16 (2)
	87	8f	97	9f	a7	af	b7	bf	[BX] + D16
mod=11	c0	c8	d0	d8	e0	e8	f0	f8	ew=AX eb=AL
	c1	c9	d1	d9	e1	e9	f1	f9	ew=CX eb=CL
	c2	ca	d2	da	e2	ea	f2	fa	ew=DX eb=DL
	c3	cb	d3	db	e3	eb	f3	fb	ew=BX eb=BL
	c4	cc	d4	dc	e4	ec	f4	fc	ew=SP eb=AH
	c5	cd	d5	dd	e5	ed	f5	fd	ew=BP eb=CH
	c6	ce	d6	de	e6	ee	f6	fe	ew=SI eb=DH
	c7	cf	d7	df	e7	ef	f7	ff	ew=DI eb=BH

(1) 8-Bit-Distanzwert wird vorzeichenbehaftet erweitert und zum Index addiert.

(2) Für Referenzen über BP ist das Segmentregister SS voreingestellt.

(3) 16-Bit-Distanzwert wird vorzeichenbehaftet zum Index addiert.

Anhang C. Übersicht hexadezimale Befehlskodierung

High \ Low	0	1	2	3	4	5	6	7
0	ADD eb,rb	ADD ew,rw	ADD rb,eb	ADD rw,ew	ADD AL,db	ADD AX,dw	PUSH ES	POP ES
1	ADC eb,rb	ADC ew,rw	ADC rb,eb	ADC rw,ew	ADC AL,db	ADC AX,dw	PUSH SS	POP SS
2	AND eb,rb	AND ew,rw	AND rb,eb	AND rw,ew	AND AL,db	AND AX,dw	Ovride ES:	DAA
3	XOR eb,rb	XOR ew,rw	XOR rb,eb	XOR rw,ew	XOR AL,db	XOR AX,dw	Ovride SS:	AAA
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI
6	PUSHA	POPA	BOUND rw,md	ARPL ew,rw				
7	JO short	JNO short	JB short	JAE short	JE short	JNE short	JBE short	JA short
8	Immed eb,db	Immed ew,dw		Immed ew,db	TEST eb,rb	TEST ew,rw	XCHG eb,rb	XCHG ew,rw
9	NOP	XCHG AX,CX	XCHG AX,DX	XCHG AX,BX	XCHG AX,SP	XCHG AX,BP	XCHG AX,SI	XCHG AX,DI
A	MOV AL,xb	MOV AX,xw	MOV xb,AL	MOV xw,AX	MOVSB	MOVSW	CMPSB	CMPSW
B	MOV AL,db	MOV CL,db	MOV DL,db	MOV BL,db	MOV AH,db	MOV CH,db	MOV DH,db	MOV BH,db
C	Shift eb,db	Shift ew,db	RET dw (near)	RET (near)	LES rw,ed	LDS rw,ed	MOV eb,db	MOV ew,dw
D	Shift eb,1	Shift ew,1	Shift eb,CL	Shift ew,CL	AAM (0A)	AAD (0A)		XLATB
E	LOOPNE short	LOOPE short	LOOP short	JCXZ short	IN AL,db	IN AX,db	OUT db,AL	OUT db,AX
F	LOCK		REPNE	REPE	HLT	CMC	Unary eb	Unary ew

Tafel 1. Operationskode 1. Byte (Fortsetzung)

High \ Low	8	9	A	B	C	D	E	F
0	OR eb,rb	OR ew,rw	OR eb,rw	OR ew,rw	OR AL,db	OR AX,dw	PUSH CS	2 Byte Opkode
1	SBB eb,rb	SBB ew,rw	SBB rb,eb	SBB rw,ew	SBB AL,db	SBB AX,dw	PUSH DS	POP DS
2	SUB eb,rb	SUB ew,rw	SUB rb,eb	SUB rw,ew	SUB AL,db	SUB AX,dw	Ovride CS:	DAS
3	CMP eb,rb	CMP ew,rw	CMP rb,eb	CMP rw,ew	CMP AL,db	CMP AX,dw	Ovride DS:	AAS
4	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6	PUSH dw	IMUL rwewdw	PUSH db	IMUL rwewdb	INSB	INSW	OUTSB	OUTSW
7	JS short	JNS short	JPE short	JPO short	JL short	JGE short	JLE short	JG short
8	MOV eb,rb	MOV ew,rw	MOV rb,eb	MOV rw,ew,	MOV ew,sr	LEA rw,m	MOV sr,ew	POP mw
9	CBW	CWD	CALL far	WAIT	PUSHF	POPF	SAHF	LAHF
A	TEST AL,db	TEST AX,dw	STOSB	STOSW	LODSB	LODSW	SCASB	SCASW
B	MOV AX,dw	MOV CX,dw	MOV DX,dw	MOV BX,dw	MOV SP,dw	MOV BP,dw	MOV SI,dw	MOV DI,dw
C	ENTER dw,db	LEAVE	RET dw (far)	RET (far)	INT3	INT db	INTO	IRET
D	80287 Befehl	80287 Befehl	80287 Befehl	80287 Befehl	80287 Befehl	80287 Befehl	80287 Befehl	80287 Befehl
E	CALL near	JMP near	JMP far	JMP short	IN AL,DX	IN AX,DX	OUT DX,AL	OUT DX,AX
F	CLC	STC	CLI	STI	CLD	STD	IncDec eb	Indrct

Tafel 2. Opkode wird durch das 2. Byte spezifiziert (1. Byte = 0FH)

Byte	00	01	02	03	04	05	06	07
	Tab.3 0F 00	Tab.3 0F 01	LAR rw,ew	LSL rw,ew			CLTS	

Tafel 3. Opkode wird durch REG-Feld (Bits 5, 4, 3) des ModRM-Bytes bestimmt (n-Format, siehe Abschn. 13.1.)

Gruppe \ REG	0	1	2	3	4	5	6	7
0F 00	SLDT ew	STR ew	LIDT ew	LTR ew	VERR ew	VERW ew		
0F 01	SGDT M	SIDT M	LGDT M	LIDT M	SMSW ew		LMSW ew	
Immed	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
Shift	ROL	ROR	RCL	RCR	SHL	SHR		SAR
Unary	TEST		NOT	NEG	MUL	IMUL	DIV	IDIV
IncDec	INC eb	DEC eb						
Indrct	INC ew	DEC ew	CALL ew	CALL ed	JMP ew	JMP ed	PUSH ew	

Legende:

Operanden	8 Bit	16 Bit	32 Bit
Direktwerte, Daten	db	dw	-
effektive Adressen (Speicher od. Register)	eb	ew	ed
Register	rb	rw	-
Segmentregister (CS,DS,ES,SS; CS != Ziel)	-	sr	-
Speicheradr. (ohne Index- od. Basisadr.)	xb	xw	-

short	8-Bit-Offset (gerechnet von der Adresse des nächsten Befehls)
near	16-Bit-Offset (gerechnet von der Adresse des nächsten Befehls)
far	32-Bit-Adresse (Reihenfolge: Offset, Segmentselektor)

Anhang D. Initialisierungsprogramm für den Mikroprozessor 80286

```
;+------------------------------------------------------------------------+
;!                                                                        !
;!              80286 - Systeminitialisierung
;!
;!   Beispielprogramm für die Initialisierung des 80286 im PVA-Mode
;!   unter möglichst vielseitiger Ausnutzung der implementierten
;!   Schutzmechanismen
;!
;+------------------------------------------------------------------------+
;
        name    Systeminitialisierung

        public  idt_desc, gdt_desc

;Definition einer Deskriptorstruktur

desc            struc
limit           dw      0               ;Segmentlimit
base_low        dw      0               ;Basisadresse Bit 0 bis 15
base_high       db      0               ;Basisadresse Bit 16 bis 23
access          db      0               ;Accessbyte
res             dw      0               ;reserviert
desc            ends

;Strukturvereinbarung für eine Taskliste

task_entry      struc                   ;Taskbeschreibung
TSS_sel         dw      ?               ;Selektor für TSS
TSS_alias       dw      ?               ;Alias-Selektor für TSS
LDT_alias       dw      ?               ;Alias-Selektor für LDT
task_entry      ends

;Fest vorgegebener Aufbau der EPROM-GDT, die nach der Umschaltung in den
;PVA-Mode in den RAM kopiert wird. Über die hier angegebenen Selektoren
;erfolgt der Zugriff auf die Systemtabellen.

gdt_alias       equ     1*size desc     ;GDT(1): RAM-Datensegment für GDT
idt_alias       equ     2*size desc     ;GDT(2): RAM-Datensegment für IDT
start_TSS_alias equ     3*size desc     ;GDT(3): RAM-Datensegment für TSS
start_task      equ     4*size desc     ;GDT(4): TSS für Starttask
start_LDT_alias equ     5*size desc     ;GDT(5): RAM-Datensegment für LDT

;Definitionen für den Zugriff auf die Bitpositionen des MSW

PE              equ     1               ;PE-Bit im MSW
MP              equ     2               ;MP-Bit im MSW
EM              equ     4               ;EM-Bit im MSW
```

```
;Definition spezieller Werte für Accessbytes in Deskriptoren

DT_ACCESS       equ     82H             ;Accessbyte für LDT
DS_ACCESS       equ     92H             ;Accessbyte für Datensegment
TSS_ACCESS      equ     81H             ;Accessbyte für leeres TSS
DPL             equ     60H             ;Maske für Privilegierungsstufe im
                                        ;Accessbyte
ACCESSED        equ     1               ;Bitmaske für "ACCESSED"
TI              equ     4               ;Bitmaske für TI-Bit im Selektor
TSS_SIZE        equ     44              ;Länge eines TSS
LDT_OFFSET      equ     42              ;Position der LDT im TSS
TIRPL_MASK      equ     7               ;Maske für TI-Bit und RPL-Feld

;+-------------------------------------+
;!                                     !
;!   Beginn des Programmkodesegments   !
;!                                     !
;+-------------------------------------+

init_code       segment         er

;Das Programmkodesegment beginnt auf der physischen Adresse OFFFE10H. Die
;ORG-Anweisung bewirkt, daß der folgende Sprungbefehl auf der physischen
;Adresse OFFFFFOH liegt, bei der die Programmabarbeitung nach einem RESET
;beginnt.

cs_offset       equ     OFE10H
                org     OFFFOH-cs_offset

;Der folgende Sprungbefehl steht jetzt auf der Startadresse OFFFFFOH.
;Das Sprungziel liegt innerhalb des aktuellen Programmkodesegments (d.h.,
;CS bleibt unverändert).

                jmp     reset_startup

;Nachfolgend wird eine temporäre GDT definiert, die nur in der Initiali-
;sierungsphase Verwendung findet. Sie wird zu Beginn der Initialisierung
;aus dem EPROM in den RAM (physische Adresse 0) kopiert.
;Dieses Segment enthält außerdem den Stackspeicherbereich und wird nach
;dem ersten Taskwechsel als TSS-Segment benutzt.

        org     0

initial_gdt     desc    <>              ;Nulldeskriptor
gdt_desc        desc    <>              ;Deskriptor für EPROM-GDT
idt_desc        desc    <>              ;Deskriptor für EPROM-IDT
temp_desc       desc    <>              ;temporärer Deskriptor

;Der folgende Deskriptor dient zum Datenzugriff auf die temporäre RAM-
;GDT selbst. Über diesen Deskriptor erfolgt auch der Stackzugriff.

temp_stack      desc    <end_gdt-initial_gdt-1,0,0,DS_ACCESS,0>
```

```
;Beim Taskwechsel zur Starttask wird der Prozessorkontext über den folgen-
;den Deskriptor abgespeichert. Der Kontext überschreibt dann den Speicher-
;bereich dieser temporären GDT.

save_tss        desc    <end_gdt-initial_gdt-1,0,0,TSS_ACCESS,0>

;Vereinbarung von 8 Worten Stackspeicherbereich

                dw      8 dup (0)

end_gdt         label   word

start_pointer   label   dword
                dw      0,start_task    ;Pointer auf Starttask

;Taskliste (Sie besteht hier nur aus einer Task.)

task_list       task_entry    <start_task,start_TSS_alias,start_LDT_alias>
                dw      0               ;Abschluß der Liste

;+-----------------------------------------------------+
;!   Beginn der Initialisierung                        !
;!   Der Mikroprozessor 80286 arbeitet hier im RA-Mode.
;+-----------------------------------------------------+

reset_startup:
        cli                             ;keine Interrupts erlaubt
        cld                             ;Autoinkrement-Mode
        xor     di,di                   ;Segmentregister mit 0
        mov     ds,di                   ;initialisieren
        mov     es,di
        mov     ss,di
        mov     sp,end_gdt-initial_gdt  ;Stackpointer  zeigt auf Ende  des
                                        ;ersten Segments

;Bildung des Korrekturfaktors zur richtigen Offsetberechnung (bedingt
;durch physische Startadresse OFFFFFOH und Startadresse 0 des Assemblers)

start   proc

        call      startl                ;Rückkehradresse im Stack  enthält
                                        ;die aktuelle Offsetkomponente
startl:
        pop     bp                      ;Rückspeichern des realen Offsets
        sub     bp,offset startl        ;Subtraktion des vom Assembler
                                        ;berechneten Offsets
```

```
;+-------------------------------------------------------------------------+
;!  Jede Datenreferenz auf den Programmkodebereich muß durch Addition  !
;!  des Index [BP] korrigiert werden, um auf die gewünschte
;!  Speicheradresse zuzugreifen!
;+-------------------------------------------------------------------------+

        lidt    initial_gdt[bp]         ;IDTR wird mit Nulldeskriptor  aus
                                        ;der  EPROM-GDT  geladen,  um  bei
                                        ;Referenzen über die IDT (z.B. bei
                                        ;Exceptions)   den   Übergang   zu
                                        ;"shut-down" zu bewirken

;Die  temporäre GDT wird aus dem EPROM in den RAM kopiert. SI enthält  den
;Pointer auf die temporäre GDT, CX die Länge (in Worten). Zieladresse  ist
;die physische Adresse 0 im RAM-Bereich.

        lea     si,initial_gdt[bp]
        mov     cx,(end_gdt-initial_gdt)/2
rep     movs    es:word ptr [di],cs:[si]

;Test auf Existenz eines Numerikcoprozessors

        fninit                          ;Initialisieren  eines evtl.  vor-
                                        ;handenen Numerikcoprozessors
        mov     bx,EM                   ;Voreinstellung: Ein Numerikcopro-
                                        ;zessor  existiert nicht. Er  wird
                                        ;emuliert.
        fstsw   ax                      ;Lesen des Coprozessorstatus
        or      al,al
        jnz     set_mode

;Numerikcoprozessor existiert

        fsetpm                          ;Numerikcoprozessor in  "protected
                                        ;mode" bringen
        mov     bx,MP                   ;Maske mit Status für  vorhandenen
                                        ;Numerikcoprozessor für MSW vorbe-
                                        ;reiten

;+-------------------------------+
;!  Umschaltung in den PVA-Mode  !
;+-------------------------------+

set_mode:
        smsw    ax                      ;AX erhält aktuelles MSW
        or      ax,PE                   ;Einblenden des PE-Bits
        or      ax,bx                   ;Numerikcoprozessorstatusbits
                                        ;einblenden
        lmsw    ax                      ;Übergang in PVA-Mode!
        jmp     $+2                     ;Befehlswarteschlange leeren!
                                        ;Es gilt: CPL = 0; CS = 0FFFE10H.
```

```
;+-----------------------------------------------------------------+
;!  Der Mikroprozessor 80286 befindet sich ab hier im PVA-Mode.   !
;+-----------------------------------------------------------------+
;
;Es  wird  jetzt das GDTR für die temporäre GDT im  RAM-Bereich  initiali-
;siert.
;
        lgdt    temp_stack[bp]
        mov     ax,temp_stack-initial_gdt
        mov     ss,ax
;
;Das  LDTR  wird zu 0 gesetzt. Damit führt jede Referenz über die  LDT  zu
;einer Exception und (auf Grund IDTR=0) zu "shut-down".
;
        xor     ax,ax
        lldt    ax

;Das Taskregister wird für den Taskwechsel in die Starttask  initialisiert
;(d.h., es zeigt auf einen RAM-Bereich, in dem ein TSS abgelegt wird).  In
;diesem Fall wird beim ersten Taskwechsel (siehe unten) der  Prozessorkon-
;text  beginnend bei der physischen Adresse 0 (Bereich der temporären  GDT
;und des Stack) in den RAM geschrieben.

        mov     ax,save_tss-initial_gdt
        ltr     ax

;Die  EPROM-GDT wird unter Nutzung des GDT-Alias-Segments in den  RAM  ko-
;piert.  Dazu muß der Deskriptor für das RAM-Datensegment in die  tempo-
;räre GDT geladen werden.

        mov     ax,gdt_desc[bp].limit           ;Länge der GDT
        cmp     ax,6*size desc-1                ;mindestens 6 GDT-Einträge
                                                ;sind notwendig
        jb      bad_gdt

        mov     bx,gdt_desc-initial_gdt         ;Alias-Datensegment für
                                                ;EPROM-GDT
        mov     si,gdt_alias                    ;Selektor für das RAM-GDT-
                                                ;Alias-Segment
        call    copy_EPROM_dt                   ;Kopieren der GDT aus  dem
                                                ;EPROM in den RAM
        mov     bx,idt_desc-initial_gdt         ;Alias-Datensegment    für
                                                ;die EPROM-IDT
        mov     si,idt_alias                    ;Selektor für das RAM-IDT-
                                                ;Alias-Segment
        call    copy_EPROM_dt                   ;Kopieren der IDT aus  dem
                                                ;EPROM in den RAM
```

```
;Nach  dem  Kopieren  der GDT in den RAM kann das GDTR auf  die  neue  GDT
;gesetzt werden.

        mov     ax,gdt_desc-initial_gdt         ;DS adressiert GDT
        mov     ds,ax
        mov     bx,gdt_alias                    ;Alias-Datensegment    für
                                                ;die RAM-GDT
        lgdt    [bx]

;Das GDTR zeigt jetzt auf die endgültige RAM-GDT.
;SS und TR zeigen noch in den RAM auf die physische Adresse 0.

;Kopieren der TSS- und LDT-Segmente aller Tasks in den RAM

        lea     bx,task_list[bp]        ;Liste der Tasks, die einzurichten
                                        ;sind
copy_task_loop:
        call    copy_tasks
        add     bx,size task_entry      ;nächster Eintrag in der Taskliste
        mov     ax,cs:[bx].tss_sel      ;Eintrag gefüllt?
        or      ax,ax
        jnz     copy_task_loop

;Einrichten des Interruptsystems durch Laden des IDTR auf kopierte IDT  im
;RAM über den Alias-Datensegmentselektor in der GDT

        mov     bx,gdt_alias            ;DS adressiert GDT
        mov     ds,bx
        mov     bx,idt_alias            ;Alias-Datensegment für RAM-IDT
        lidt    [bx]

;+-------------------------------------------------+
;!  Taskwechsel zur Starttask (indirekter JMP zu TSS)
;+-------------------------------------------------+

        jmp     start_pointer[bp]       ;Start der ersten Task!

;Das  erste  TSS (physische Adresse 0 im RAM-Bereich) wird jetzt  mit  dem
;aktuellen  CPU-Kontext überschrieben. Der neue Prozessorkontext wird  aus
;dem vorbereiteten TSS in die CPU geladen.

;=============================================================================

bad_gdt:
        hlt                             ;GDT ist nicht groß genug

start   endp

bad_tss:
        hlt                             ;ungültiges TSS
```

```
;+-------------------------------------------------------------------------+
;!  Unterprogramm: copy_tasks                                              !
;!    Das im EPROM vorbereitete TSS und eine eventuell vorhandene  LDT
;!    werden  in  den RAM kopiert. Die Task wird durch  CS:BX  in  der
;!    Taskliste spezifiziert.
;+-------------------------------------------------------------------------+

copy_tasks      proc

        mov     si,gdt_alias            ;DS adressiert GDT
        mov     ds,si
        mov     si,cs:[bx].tss_alias    ;ES adressiert TSS-Alias-Segment
        mov     es,si
        lsl     ax,si                   ;Länge des TSS-Alias-Segments
        mov     si,cs:[bx].tss_sel      ;TSS-Selektor
        lar     dx,si                   ;Zugriffsrechte testen
        jnz     bad_tss                 ;Sprung, wenn Referenz unzulässig

        mov     dl,dh                   ;TSS-Accessbyte merken
        and     dh,not DPL              ;Ignorieren der DPL
        cmp     dh,TSS_ACCESS           ;Test auf gültiges TSS
        jnz     bad_tss                 ;Sprung, wenn kein TSS-Segment

        lsl     cx,si                   ;Länge des EPROM-TSS
        cmp     cx,TSS_SIZE-1           ;Test auf Einhaltung  TSS-Mindest-
                                        ;länge
        jb      bad_tss

;Vorbereitung des Transfers des TSS aus dem EPROM in den RAM
;Da für den Transfer der Datenzugriff auf das EPROM-TSS notwendig ist, muß
;das Accessbyte des TSS-Deskriptors modifiziert werden.

        mov     [si].access,DS_ACCESS   ;Datenzugriff auf TSS ermöglichen
        mov     ds,si
        call    copy_with_fill          ;Kopieren des TSS aus EPROM in RAM

;In der GDT müssen die Limit- und Basiswerte des TSS auf die  entsprechen-
;den Werte des RAM-TSS gesetzt werden.

        mov     ax,gdt_alias            ;DS und ES adressieren GDT
        mov     ds,ax
        mov     es,ax
        mov     si,cs:[bx].tss_alias    ;Quelle: Datensegmentdeskriptor
                                        ;        des TSS-Alias-Segments
        mov     di,cs:[bx].tss_sel      ;Ziel: TSS-Deskriptor

        movsw                           ;Limit übertragen
        movsw                           ;Basisadresse Bit 0 bis 15
        lodsw                           ;Basisadresse Bit 16 bis 23 in AL
        mov     ah,dl                   ;TSS-Accessbyte einblenden
        stosw                           ;High-Teil Basisadresse und
                                        ;Accessbyte
        movsw                           ;reserviertes Wort (null)
```

```
;Test, ob eine gültige LDT für die Starttask angegeben ist.
;DS wird zur Adressierung des TSS-Segments mit dem TSS-Alias-Segmentselek-
;tor  geladen. SI erhält den LDT-Selektor aus dem TSS. Das TI-Bit und  das
;RPL-Feld werden ausgeblendet.

        mov     ds,cs:[bx].tss_alias
        mov     si,ds:word ptr LDT_OFFSET
        and     si,not TIRPL_MASK
        jz      no_ldt                  ;Sprung, wenn LDT-Selektor
                                        ;ungültig

        push    si                      ;LDT-Selektor retten
        lar     dx,si                   ;Test auf Zugreifbarkeit
        jnz     bad_ldt

;Vorbereitete LDT aus dem EPROM in den RAM kopieren

        mov     dl,dh                   ;Accessbyte des LDT-Deskriptors
                                        ;merken
        and     dh,not DPL              ;Ignorieren der DPL
        cmp     dh,DT_ACCESS            ;Test auf LDT-Deskriptor
        jne     bad_ldt                 ;Sprung, wenn LDT-Deskriptor
                                        ;ungültig

;ES  adressiert  die aktuelle GDT über das  GDT-Alias-Segment.  Durch  die
;folgende  Modifikation des LDT-Deksriptors wird der Datenzugriff für  den
;Kopiervorgang der LDT erlaubt.

        mov     es:[si].access,DS_ACCESS

;DS  wird  für den Zugriff auf die EPROM-LDT geladen.  Die  zu  kopierende
;Länge wird aus dem LDT-Deskriptor entnommen.
;Das  Unterprogramm test_dt_limit führt einen Plausibilitätstest  für  die
;Länge der Deskriptortabelle durch.

        mov     ds,si                   ;Alias-Datensegment der EPORM-LDT
        lsl     ax,si                   ;Länge der EPROM-LDT
        call    test_dt_limit           ;Plausibilitätstest
        mov     cx,ax                   ;Zwischenspeicher

;Überprüfen des LDT-Alias-Segments und Kopieren der LDT

        mov     si,cs:[bx].ldt_alias    ;LDT-Alias-Selektor
        mov     es,si                   ;ES adressiert EPROM-TSS
        lsl     ax,si                   ;Länge des TSS-Alias-Datensegments
        call    test_dt_limit           ;Plausibilitätstest
        call    copy_with_fill          ;Kopiervorgang EPROM-LDT in RAM
```

```
;In  der GDT müssen die LDT-Länge und die LDT-Basisadresse auf  die  Werte
;der RAM-LDT gesetzt werden

        mov     si,cs:[bx].ldt_alias    ;Quelle: LDT-Alias-Selektor
        pop     di                      ;Ziel  : LDT-Selektor
        mov     ax,gdt_alias            ;DS und ES adressieren GDT
        mov     ds,ax
        mov     es,ax
        movsw                           ;Länge der RAM-LDT
        movsw                           ;Basisadresse Bit 0 bis 15
        lodsw                           ;Basisadresse Bit 16 bis 23
        mov     ah,dl                   ;Einblenden LDT-Accessbyte
        stosw                           ;Basisadresse High-Teil und
                                        ;Accessbyte
        movsw                           ;reserviertes Wort
;
no_ldt:
        ret

bad_ldt:
        hlt                             ;ungültige LDT

copy_tasks      endp

;+-------------------------------------------------------------------+
;!  Unterprogramm: test_dt_limit                                     !
;!    Überprüfen der Länge einer Deskriptortabelle
;!    Die Länge+1 muß durch 8 teilbar sein, d.h., die Bitpositionen
;!    0 bis 2 müssen den Wert 111 besitzen.
;!    Die Länge wird in AX übergeben.
;+-------------------------------------------------------------------+

test_dt_limit   proc

        push    ax                      ;Länge retten
        and     al,7                    ;Bit 0 bis 2 ausblenden
        cmp     al,7
        pop     ax
        jne     bad_dt_limit
        ret

bad_dt_limit:
        hlt                             ;Abbruch

test_dt_limit   endp
```

```
;+-------------------------------------------------------------------------+
;!  Unterprogramm: copy_EPROM_dt                                           !
;!    Kopieren einer Deskriptortabelle aus dem EPROM in den RAM            !
;!    BX: Selektor für Alias-Datensegment der EPROM-DT in der              !
;!        temporären GDT                                                   !
;!    SI: Selektor für Alias-Datensegment der RAM-DT                       !
;!    Ungültige Deskriptoren oder Deskriptorlängen führen zu "shut-down"   !
;+-------------------------------------------------------------------------+

copy_EPROM_dt   proc

;Zuerst  wird der Alias-Datensegmentdeskriptor der EPROM-DT in die  tempo-
;räre GDT kopiert (temp_desc)

        mov     ax,ss                   ;ES adressiert temporäre GDT
        mov     es,ax

;Der Datenzugriff auf die EPROM-DT wird durch Modifikation des Accessbytes
;des entsprechenden Deskriptors in der temporären GDT ermöglicht.

        mov     es:[bx].access,DS_ACCESS
        mov     es:[bx].res,0           ;reserviertes Wort garantieren
        lsl     ax,bx                   ;Länge der EPROM-DT
        mov     cx,ax                   ;Zwischenspeicher
        call    test_dt_limit           ;Plausibilitätstest

;DS wird jetzt zur Adressierung der EPROM-DT geladen. Zieladresse ist  der
;temporäre Deskriptor. Der Selektor in DI wird später zur Adressierung des
;Zielsegments noch benötigt.

        mov     di,gdt_desc-initial_gdt
        mov     ds,di
        mov     di,temp_desc-initial_gdt
        push    di

;Der  Deskriptorinhalt  wird  aus  der  EPROM-DT  in  die  temporäre  GDT
;übertragen.

        lodsw                           ;Alias-Segmentlänge
        call    test_dt_limit           ;Plausibilitätstest
        stosw                           ;Länge kopieren
        movsw                           ;Rest des Deskriptors kopieren
        movsw
        movsw

;Nach  Anlegen des Zieldeskriptors in der temporären GDT werden  die  Seg-
;mentregister  für den Zeichenkettentransfer gesetzt. ES  adressiert  dann
;den  RAM-Bereich, in den die EPROM-DT kopiert werden soll. DS  adressiert
;die EPROM-DT über das Alias-Datensegment.

        pop     es                      ;Zielselektor (temp_desc)
        mov     ds,bx                   ;Quellselektor (Argument beim  UP-
                                        ;Aufruf)
```

```
;CX enthält jetzt die Anzahl der zu kopierenden Bytes. Die Differenz AX-CX
;gibt die Länge (in Bytes) an, die im Alias-Datensegment mit Nullen aufzu-
;füllen ist.
;Übergang in das Unterprogramm copy_with_fill

copy_EPROM_dt   endp

;+-----------------------------------------------------------------------+
;!  Unterprogramm: copy_with_fill                                        !
;!    Kopieren eines Segments der Länge CX von DS nach ES.
;!    Wenn das Alias-Datensegment größer ist als das Quellsegment,
;!    wird der Rest mit Nullen aufgefüllt.
;!    CX:    Anzahl der zu kopierenden Bytes
;!    AX-CX: Anzahl aufzufüllender Nullen (in Bytes)
;+-----------------------------------------------------------------------+

copy_with_fill  proc

        xor     si,si                   ;Segmentanfangsadresse null
        xor     di,di
        sub     ax,cx                   ;AX := Anzahl aufzufüllender
                                        ;Nullen
        add     cx,1                    ;Anzahl = Länge + 1
        rcr     cx,1                    ;Korrektur für Wortoperationen
                                        ;CF = 1, wenn Anzahl ungerade
rep     movsw                           ;Kopiervorgang
        xchg    ax,cx                   ;CX := Anzahl aufzufüllender
                                        ;Nullen, AX := 0
        jnc     even_copy               ;Sprung, wenn Anzahl gerade ist

        movsb                           ;Kopieren des ungeraden Bytes
        or      cx,cx                   ;Alias-Datensegment gleich Quell-
                                        ;segment?
        jz      exit_copy
        stosb                           ;für Wortzugriff auf gerade Adres-
                                        ;sen vorbereiten
        dec     cx                      ;Korrektur
;
even_copy:
        shr     cx,1                    ;Wortanzahl für Füllvorgang
                                        ;CF = 1, wenn Anzahl ungerade
rep     stosw                           ;Löschen der unbenutzten Worte am
                                        ;Segmentende
        jnc     exit_copy
        stosb                           ;Löschen des letzten ungeraden
                                        ;Bytes
exit_copy:
        ret

copy_with_fill  endp

init_code       ends
        end
```

Literaturverzeichnis

/1/ Blomeyer-Bartenstein, H. P.: Microcomputertechnik. München: Hofacker Verlag 1977

/2/ Osborn, A.: Einführung in die Mikrocomputer-Technik. München: te-wi Verlag 1977

/3/ Claßen, L.; Oefler, U.: Wissenspeicher Mikroprozessorprogrammierung. 4. Aufl. Berlin: VEB Verlag Technik 1989

/4/ Kieser, H.; Meder, M.: Mikroprozessortechnik. 3. Aufl. Berlin: VEB Verlag Technik 1985

/5/ Kieser, H.; Bankel, M.:Einchipmikrorechner. Berlin: VEB Verlag Technik 1986

/6/ Stewen, L.: Mikroprozessortechnik. Heidelberg: Hüthig Verlag 1988

/7/ Alexy, G.N.; Rector, R.: The 8086 Book. Berkeley, Calif.: Osborne-McGraw-Hill 1980

/8/ Bonitz, J.: Der 16-Bit-Mikroprozessor des ESER-PC. Berlin: VEB Verlag Technik 1989

/9/ Norten, P.: Programmierhandbuch für den IBM PC. Braunschweig: Vieweg & Sohn Verlagsgesellschaft 1986

/10/ DCP Software Dokumentation / Anwendungsbeschreibung Hard- und Software. Firmenschrift. Karl-Marx-Stadt: VEB Robotron-Buchungsmaschinenwerk 1987

/11/ iAPX 86 User's Manual. Firmenschrift. Santa Clara, Calif.: Intel Corp. 1981

/12/ iAPX 286 Hardware Reference Manual. Firmenschrift. Santa Clara, Calif.: Intel Corp. 1983

/13/ 80386 Hardware Reference Manual. Firmenschrift. Santa Clara, Calif.: Intel Corp. 1986

/14/ Introduction to the iAPX 286. Firmenschrift. Santa Clara, Calif.: Intel Corp. 1982

/15/ iAPX 286 Programmer's Reference Manual. Firmenschrift. Santa Clara, Calif.: Intel Corp. 1985

/16/ iAPX 286 Operating System Writer's Guide. Firmenschrift. Santa Clara, Calif.: Intel Corp. 1983

/17/ Vieillefond, C.: Programmierung des 80286. Düsseldorf: Sybex Verlag 1987

/18/ Thies, K.-D.: Die innovativen 80286/80386 Architekturen. München: te-wi Verlag 1986

/19/ Strauss, E.: Inside the 80286. New York: Prentice Hall Press 1986

/20/ Geyer, J.: Schneller Prozessor arbeitet mit virtuellem Adreßbereich. Elektronik 1982, H. 5, S. 59-68

/21/ Multibus I Architecture Reference Book. Firmenschrift. Santa Clara, Calif.: Intel Corp. 1983

/22/ Multibus IEEE P796 Standardentwurf. IEC 47B (secretariat) 19

/23/ Thies, K.-D.: Die 8087/80287 Numerischen Prozessor-Erweiterungen für 8086/80286 Systeme. München: te-wi Verlag 1985

Sachwörterverzeichnis

Der Mikroprozessor 80286 hat international eine außerordentliche Bedeutung erlangt, die neben seinen hervorragenden technischen Leistungsparametern auf der Softwarekompatibilität unter den Betriebssystemen MS-DOS, OS/2 und UNIX beruht.
In diesem Buch werden der Aufbau, die innere Struktur, die verschiedenen Betriebsarten und vor allem der ganze Komplex der Programmierung des 80286 unter dem Blickwinkel der Software dargestellt.

ISBN 3-341-00867-5