KAPITEL 3

Schutz

3.1 Schutztypen

Der 80286 kennt drei wesentliche Anwendungsfälle des Schutzes:

- Die Isolierung von System-Software gegen Benutzer-Software.
- Die Isolierung mehrerer Benutzer gegeneinander (Intertask-Schutz).
- Die Überprüfung von Datentypen.

Der erste Fall ist besonders in einer Umgebung wichtig, in der mehrere Benutzer gleichzeitig unterstützt werden. In dieser Umgebung sind Software-Module, die sich in einer "Überwacher"-Ebene befinden, vor Modulen in der Benutzer-Ebene geschützt.
Schränkt man die Adressierfähigkeit von Benutzer-Software ein, kann das Betriebssystem die System-Ressourcen ungestört überwachen und steuern, was insbesondere in Multi-User-, Multi-Tasking- und verteilten Verarbeitungs-Systemen eine Grundvoraussetzung bildet.

Der zweite Fall des Schutzes betrifft die Isolierung von Benutzern. Ohne diese Isolierung könnte ein Fehler in einem Benutzer-Programm die Operationen eines anderen fehlerfreien Benutzer-Programms beeinflussen. Solche Interaktionen sind schwierig zu diagnostizieren und zu beseitigen. Eine wesentliche Verbesserung der Zuverlässigkeit von Benutzer-Programmen ist nur dann möglich, wenn die Benutzer voneinander isoliert sind.

Im dritten Fall des Schutzes gewährleistet die 80286-CPU, daß innerhalb eines System-oder Benutzer-Programms alle Code- und Daten-Segmente richtig benützt werden. So ist es nicht möglich, daß Daten ausgeführt und Code modifiziert werden. Zusätzlich müssen sich "Offsets" innerhalb der definierten Segmentobergrenzen aufhalten usw. Solche Überprüfungen werden bei jedem Speicherzugriff durchgeführt.

3.2 Schutz-Implementierung

Die Schutzhardware des 80286 stellt den Umgang mit dem Speicher, den I/O-Geräten und den Befehlen unter Regeln. Praktisch ist zwar die Anzahl der möglichen Interaktionen zwischen Befehlen, dem Speicher und den I/O-Geräten unbegrenzt; durch den Schutzmechanismus wird sie aber auf eine bestimmte Teilmenge eingeschränkt.

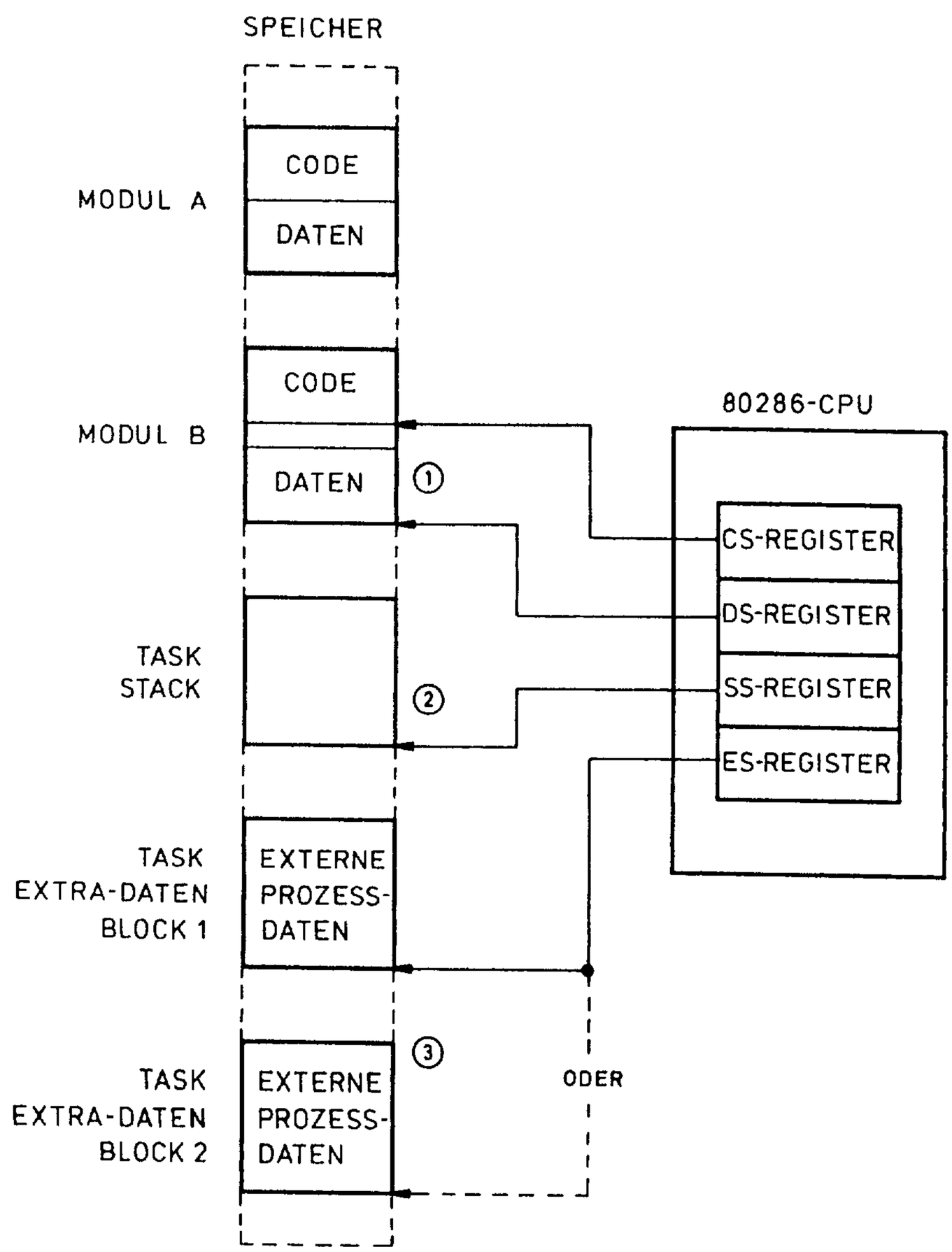
Nur Operationen aus dieser Teilmenge sind korrekt. Jede Operation, die nicht zu dieser Teilmenge gehört, wird von der Schutzhardware festgestellt und als Schutzverletzung signalisiert.

Um den Schutzmechanismus des 80286 zu verstehen, sollen zunächst die Grundbegriffe "Segmente" und "Tasks" erläutert werden.

Segmente sind die kleinsten zusammenhängenden Regionen des Speichers mit eigenen Schutz-Attributen. Modulare Programmierung ergibt automatisch Segmente, deren Inhalte als Einheit behandelt werden können.

Im Kapitel "Speicherorganisation" wurde bereits darauf hingewiesen, daß Segmente den funktionellen Aufbau eines Programms wiedergeben. So existieren z. B. für ein Modul ein Code-Segment, ein Daten-Segment und für die "Task" ein Stack-Segment.

Das folgende Bild illustriert die Adressierung von Segmenten eines Moduls innerhalb einer "Task".



Es ist ersichtlich, daß 2 Module (A und B) existieren, die je ein Code- und Daten-Segment enthalten. Die augenblicklich adressierbaren Segmente in Modul B sind durch einen entsprechenden CS-Selektor und den zugehörigen DS-Selektor ausgewählt ①.

Da sich Modul A und B in der augenblicklich aktiven "Task" aufhalten, beziehen sie sich auf ein gemeinsames Stack-Segment, das durch einen SS-Selektor ausgewählt ist ②.

Das Bild zeigt weiterhin zwei zusätzliche Datenblöcke, sogenannte Extra-Daten-Segmente, die nur innerhalb der aktiven "Task" verfügbar sind und z. B. externe Prozeßdaten enthalten. Die Extra-Daten-Segmente können jedem Modul zugewiesen werden. Zu diesem Zweck wird z. B. der ES-Selektor benützt, um eines dieser Segmente auszuwählen ③.

Die Speicher-Segmentierung des 80286 wurde für eine optimale Ausführung des Codes unabhängiger Module konzipiert.

Jedes Segment wird durch einen speicher-residenten Deskriptor angesprochen. Zugriffe auf Segmente, die ***nicht*** der Definition des Deskriptors entsprechen, können deswegen von der Schutzhardware verhindert werden. Die Fähigkeit, Zugriffe auf Segmente überwachen zu können, ist Voraussetzung des Softwareschutzes, der durch:

- separate Adreßräume
- Privileg-Mechanismen

verwirklicht wird.

Separate Adreßräume trennen die Anwender-Programme voneinander; Privileg-Mechanismen isolieren System-Sofware von Anwender-Software.
Software, die das gesamte System steuert, muß einen hohen Vertrauensgrad haben und wird sich typischerweise in der privilegiertesten Ebene aufhalten.
Eine "normale" Anwender-Software hat mit diesen Steuer-Mechanismen nichts zu tun, die nur dann zur Anwendung kommen, wenn "Task"-Wechsel erfolgen oder Betriebssystem-Routinen aufgerufen werden müssen.

Einen wichtigen Unterschied gibt es zwischen "Tasks" und Programmen. Programme sind ***statisch*** und bestehen aus einem festen Satz von Code- und Daten-Segmenten. Dabei hält sich jeder dieser Segmente in einer bestimmten Privileg-Ebene auf.

Im Gegensatz dazu sind "Tasks" ***dynamisch***. Sie führen ein oder mehrere Programme aus, wenn sich diese im Adreßraum der "Task" aufhalten. Zusätzlich bestimmen die von der Schutzart her vorgegebenen Regeln, wann ein Programm ausgeführt werden darf, und nach der Ausführung, was das Programm tun darf.

3.3 Memory Management und Schutz

Die Schutzhardware des 80286 ist eng mit der Memory Management-Hardware verbunden. Schutz-Attribute werden zusammen mit der Memory Management-Information (wie Basisadresse und Limit) im Segment-Deskriptor gespeichert.

Die Schutz-Attribute eines Segments werden spezifiziert, wenn durch den "System-Builder" BLD286 das Segment eingerichtet wird. Diese Attribute kennzeichnen unter anderem die Privileg-Ebene des Segments und definieren den Segment-Typ (z. B. Code-Segment, Daten-Segment usw.).

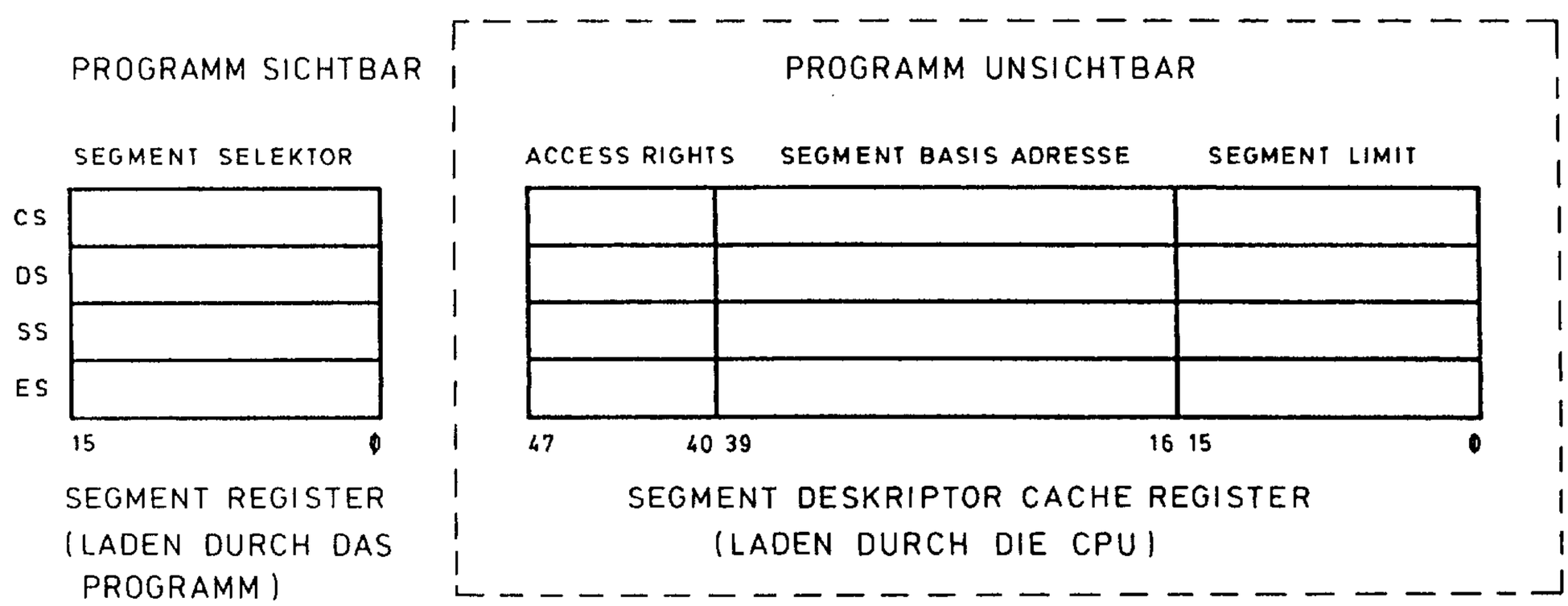
Somit besteht die Schutzinformation für ein Segment aus:

- Segment-Typ
- Privileg-Ebene
- Segment-Größe.

Segment-Typ und Privileg-Ebene sind im Access-Byte des Segment-Deskriptors codiert. Die Segment-Größe ist im Limit-Feld gespeichert.

Geändert werden Schutzinformationen nur, wenn die Selektor-Komponente eines Segment-Registers vom Programm geladen wird. Immer dann initialisiert die CPU den unsichtbaren Teil des Segment-Registers mit einem neuen Deskriptor. Da sich dieses Deskriptor-Register während der Zugriffe auf die Segment-Eintragungen für den Programmierer "versteckt" hält, wird es als Segment-Deskriptor-***Cache***-Register bezeichnet.

Das folgende Bild soll dies verdeutlichen.



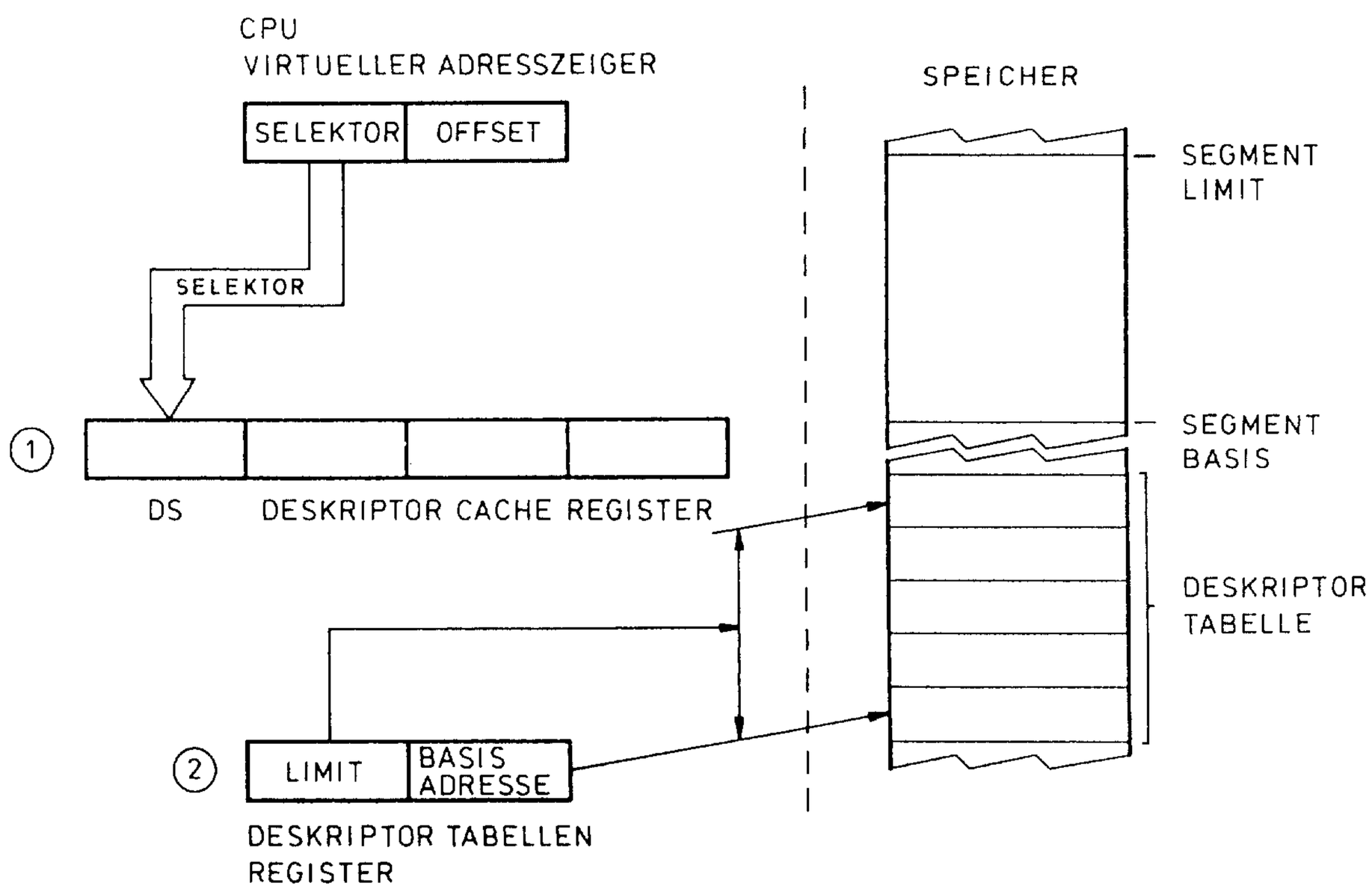
Auf die Schutzdaten Segment-Typ, Privileg-Ebene und Segment-Größe bezieht sich die CPU ***zweimal,*** nämlich

- bei jedem Laden eines Segment-Registers und
- bei jedem Zugriff auf Eintragungen im selektierten Segment.

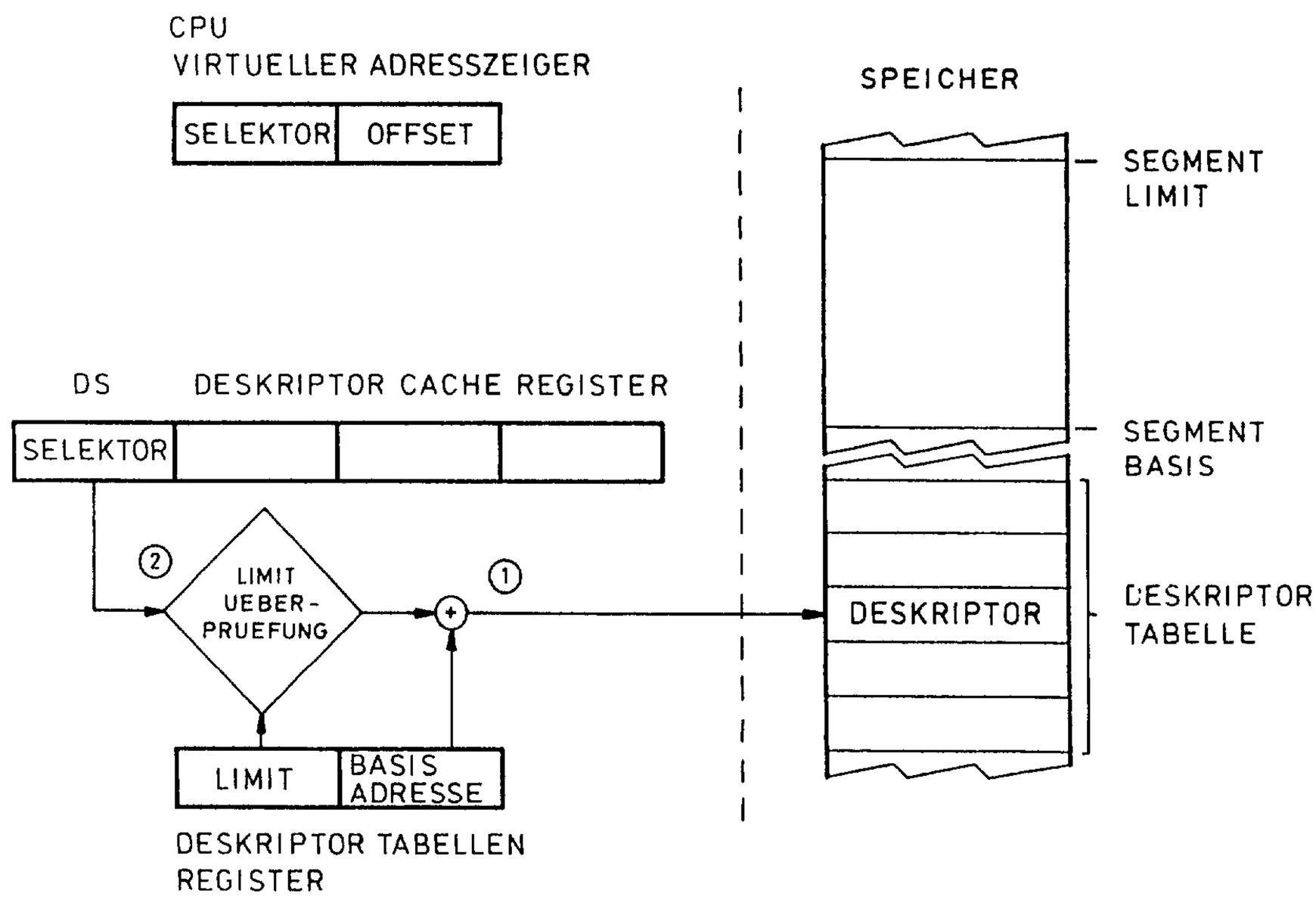
Die Hardware nimmt dabei verschiedene Überprüfungen vor, die von den Schutzregeln vorgegeben sind.

Anhand der nachstehenden Bilderfolge sollen diese Überprüfungen erläutert werden.

1) Zunächst lädt das Programm einen Selektor in eines der Segment-Register. Im Beispiel ist dies das Register DS ①. Die Position und die Größe der Deskriptor-Tabelle sei entweder durch das "Global Descriptor Table Register" GDTR oder das "Local Deskriptor Table Register" LDTR gekennzeichnet ②.

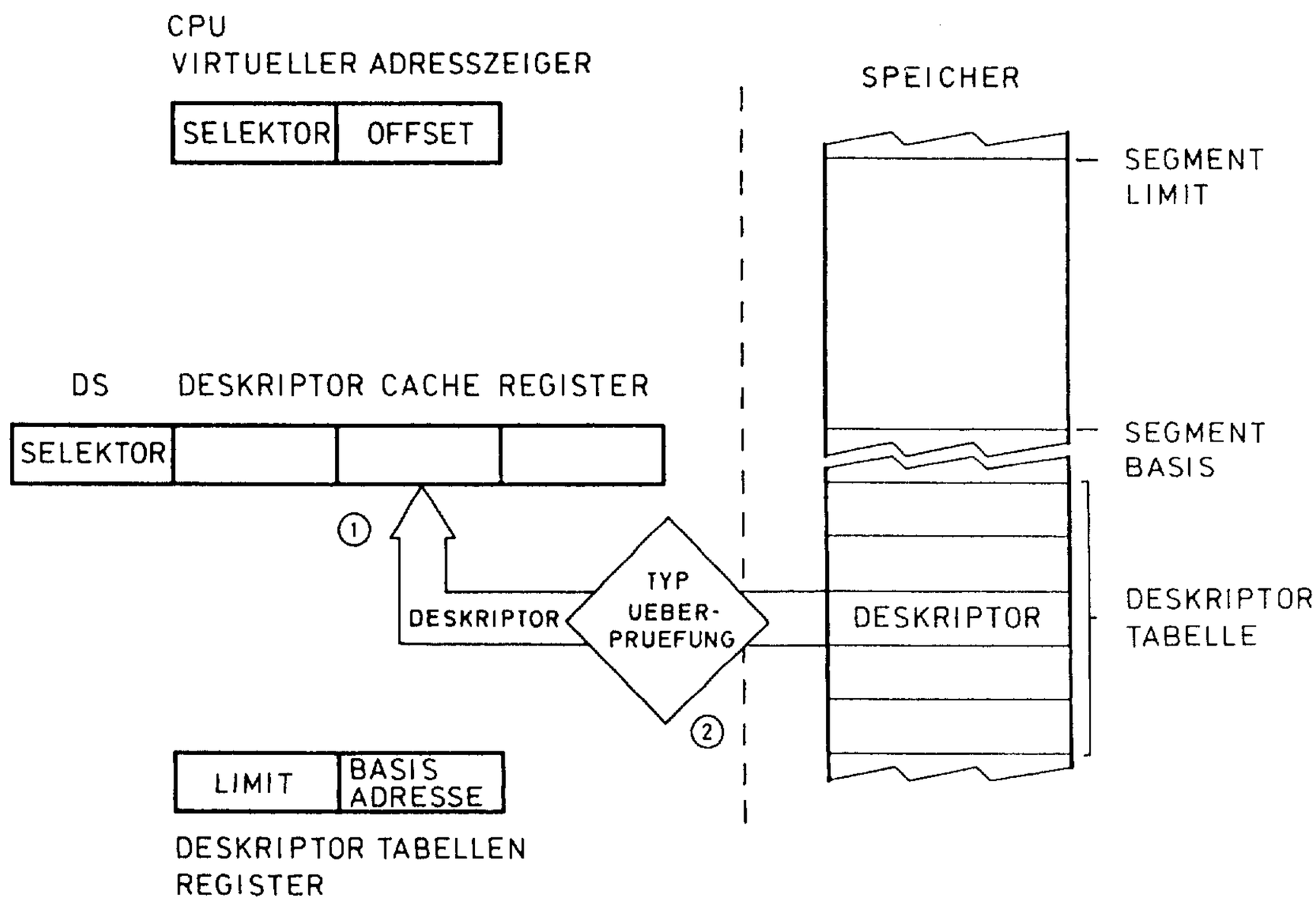


2) Ist der Selektor geladen, addiert die CPU den Selektor-Index zur Basisadresse der Deskriptor-Tabelle (GDT oder LDT), um einen Deskriptor auszuwählen ①.



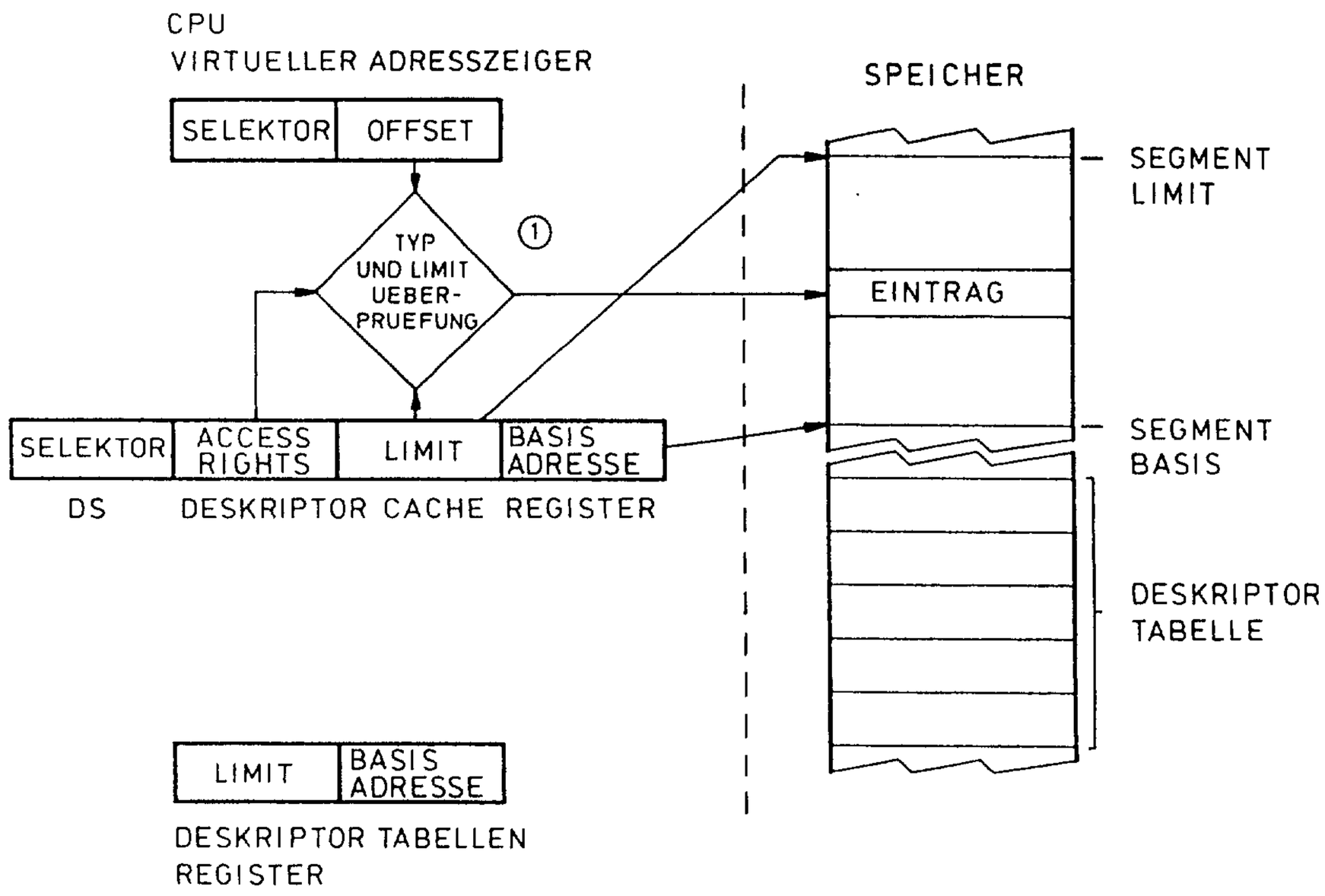
- Alle Deskriptor-Tabellen haben eine spezifische Obergrenze (Limit). Sie wird von der Schutzhardware benützt, um die Adressraum-Trennung der "Tasks" sicherzustellen.
- Jeder Versuch des Programms, ein Segment-Register oder auch das "Local Deskriptor Table Register" LDTR mit einem Selektor zu laden, der sich auf einen Deskriptor ***außerhalb*** der korrespondierenden Tabellen-Obergrenze bezieht, führt zwangsläufig zur Unterbrechung des Programms.
  Eine solche Situation wird als Ausnahme bzw. "Exception" bezeichnet.
  Die CPU nimmt also bei jedem Zugriff auf eine Deskriptor-Tabelle eine "Limit"-Überprüfung vor ②.
- Es sei darauf hingewiesen, daß die GDT oder LDT nur gültige Deskriptoren enthalten dürfen.
  Allerdings sind Löcher bzw. "leere" Deskriptoren erlaubt. "Leere" Deskriptoren bieten die Möglichkeit, Segmente dynamisch zuzuweisen oder auch zu tilgen, ohne die Größen der GDT oder LDT ändern zu müssen.
  Der Versuch, einen Selektor zu laden, der sich auf einen "leeren" Deskriptor bezieht, führt aber zur Unterbrechung des Programms.

3) Hat der Zugriff auf einen Deskriptor zu keiner "Exception" geführt, kopiert die CPU den Deskriptor in das DS-Cache-Register ①.



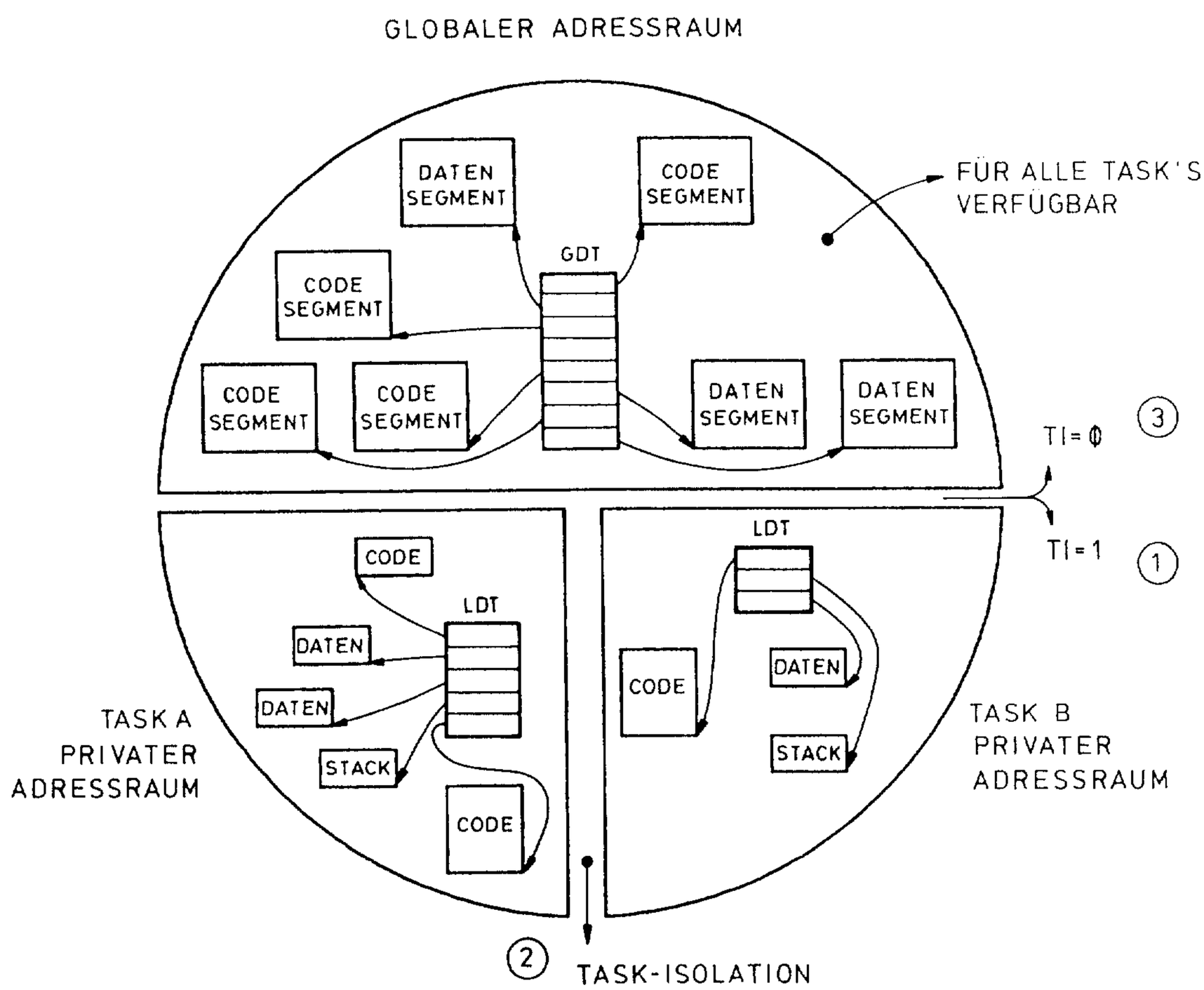
- Jedesmal, wenn der unsichtbare Teil eines Segment-Registers mit einem Deskriptor geladen wird, überprüft die Schutzhardware, ob dieser Deskriptor zum Typ des Segment-Registers paßt ②.
- So ist es z. B. nicht möglich, daß der Deskriptor eines "Read Only"-Segments in das SS-Cache-Register geladen wird, da ein in diesem Register enthaltener Deskriptor ein Stack-Segment spezifiziert, das auch beschreibbar sein muß (Typ "Read/Write").
- Die gleiche Typüberprüfung wird auch beim Laden des "Local Descriptor Table Registers" LDTR durchgeführt. Hier darf nur ein LDT-Deskriptor in den unsichtbaren Teil des Registers kopiert werden.
- Jede Verletzung dieser Schutzregeln stellt eine "Exception" dar und führt zur Unterbrechung des Programms.

4) Ist ein Deskriptor erfolgreich geladen worden, benützt nun die Schutzhardware die Deskriptor-Informationen, um den Typ und die Obergrenze des ausgewählten Segments zu überprüfen ①.



- Während die Adreßeinheit den Offset auf die Basisadresse des Segments addiert, prüft gleichzeitig die Schutzhardware, ob dieser Offset nicht größer als die Segmentlänge ist. Sie verhindert dadurch, daß ein Zugriff über das Ende des Segments hinaus und damit eventuell auf ein anderes Segment erfolgt.
- Parallel dazu wird überprüft, ob der Zugriff mit dem Typ des Segments vereinbar ist. Dadurch wird z. B. verhindert, daß in ein "Execute Only"- oder "Read Only"-Segment geschrieben oder aus einem "Execute Only"-Segment gelesen wird.
- Jede Verletzung dieser Schutzregeln stellt eine "Exception" dar und führt zur Unterbrechung des Programms.

Die erläuterten Überprüfungen, die bei jedem Speicherzugriff durchgeführt werden, gehören zum einfacheren Typ des im 80286 implementierten Schutzmechanismus. Durch diese Überprüfungen kann sichergestellt werden, daß mehrere Benutzer bzw. "Tasks" voreinander geschützt sind.
Dies soll das folgende Bild illustrieren.



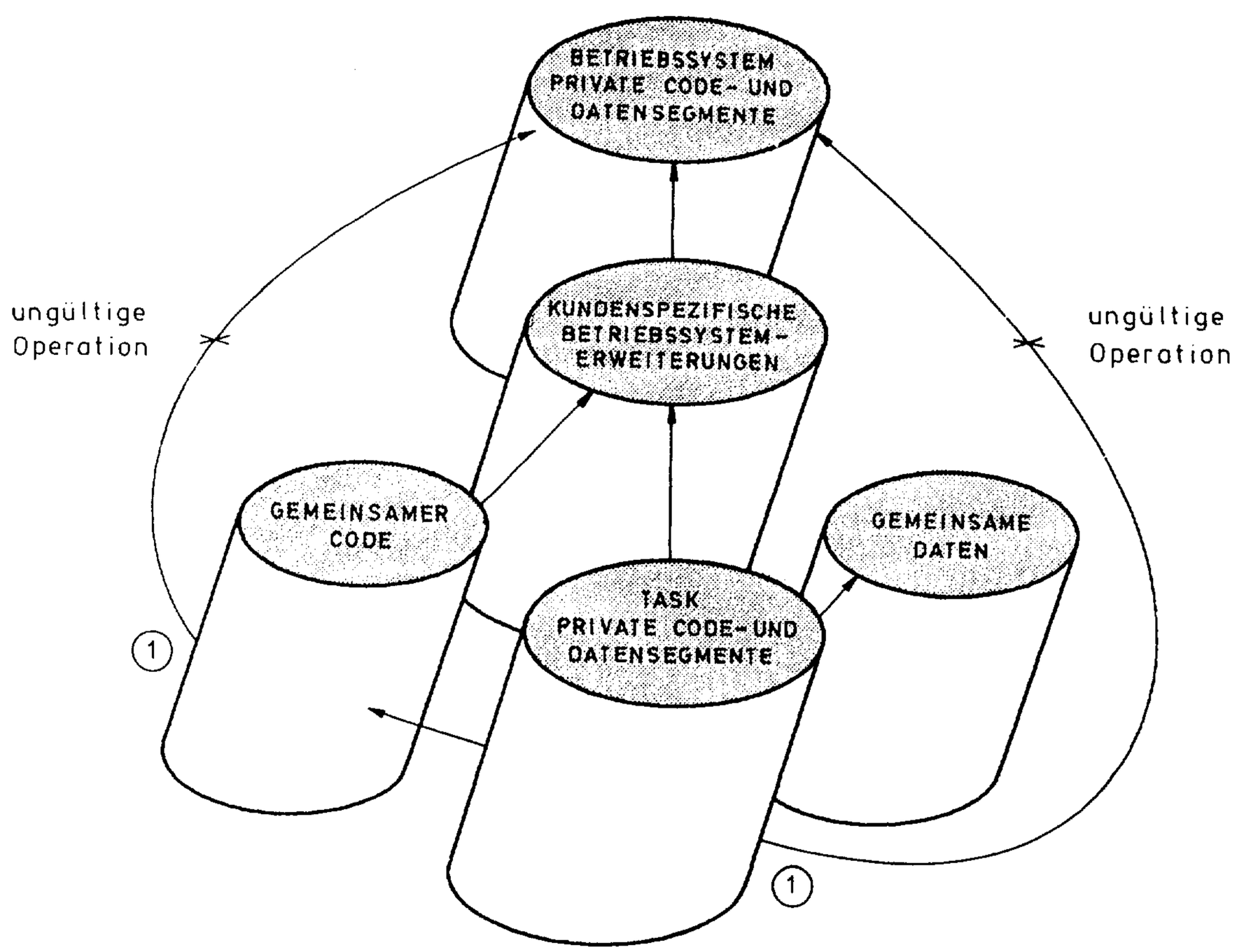
Der gültige Adressraum einer "Task" wird durch ihre LDT und die GDT spezifiziert. Während die LDT den privaten Adressraum einer "Task" festlegt, definiert die GDT einen Adressraum, der für alle "Tasks" verfügbar ist.
Bei jedem Zugriff auf einen Deskriptor in der LDT ("Table Indikator" = 1) ① stellt die Schutzhardware fest, ob sich der Deskriptor jenseits der adressierbaren LDT befindet (Index-Limit-Überprüfung). Wenn ja, käme dies einem Zugriff auf einen Adressraum gleich, der außerhalb der aktiven "Task" liegt. Eine Programmunterbrechung verhindert dies und garantiert damit die Isolierung der "Tasks" ②.

Die gleiche "Limit"-Überprüfung wird bei einem Zugriff auf einen Deskriptor in der GDT ("Table Indikator" = 0) durchgeführt ③. Auch hier verhindert eine Programmunterbrechung, daß ein Zugriff auf eine Speicher-Position erfolgt, die außerhalb des globalen Adressraums liegt.

Im gültigen Adressraum einer "Task" halten sich die Segmente auf, die mit bestimmten Zugriffsrechten ausgestattet sind. Bei jedem Zugriff auf einen Eintrag überprüft die Schutzhardware, ob eine Verletzung dieser Zugriffsrechte vorliegt, und stellt zusätzlich fest, ob der Eintrag ungültig ist, sich also jenseits des adressierbaren Segments befindet (Offset-Limit-Überprüfung).
Ist das Ergebnis der Überprüfungen positiv, löst die Schutzhardware in beiden Fällen eine Programmunterbrechung aus.

3.4 Privilegebenen und Schutz

Um innerhalb der aktiven "Task" Betriebssystem-Software von Anwender-Software zu isolieren, ist eine weitere Dimension des Schutzes notwendig.
Das folgende Bild zeigt das Modell einer typischen Task-Umgebung zusammen mit dem erforderlichen Zugriffsschutz.



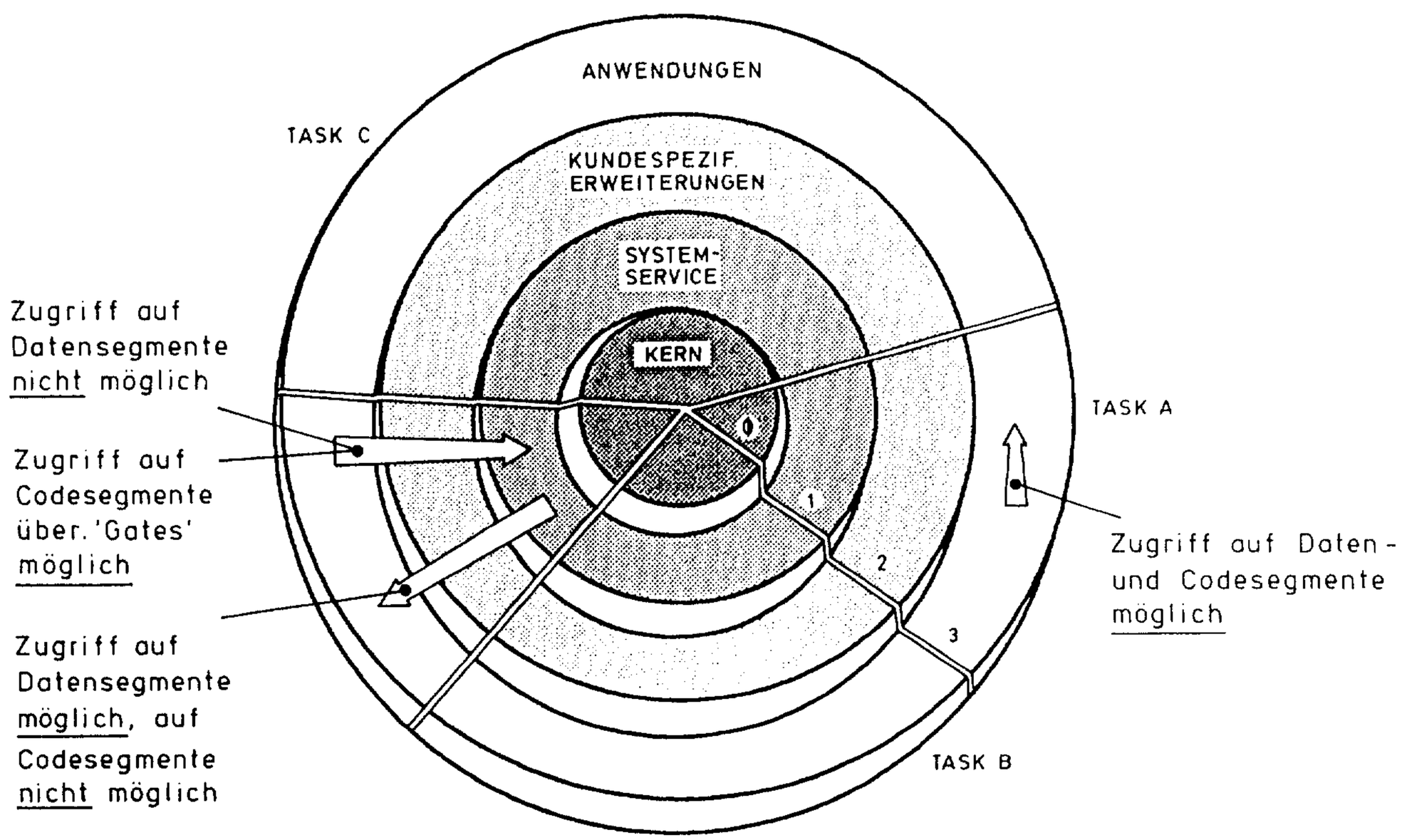
Im lokalen Adressraum der "Task" liegen deren private Code- und Datensegmente.

Im globalen Adressraum der "Task" liegen

- gemeinsame Code- und Datensegmente, die für alle "Tasks" verfügbar sind,
- kundenspezifische Betriebssystem-Erweiterungen, die für alle Benutzer verfügbar sind, und
- private Code- und Datensegmente des Betriebssystems.

Die Betriebssystem-Software ist nur dann vor Anwender-Software zu schützen, wenn direkte Zugriffe einer "Task" oder eines globalen Codes auf Routinen des Betriebssystems als ungültige Operationen erkannt werden ①.
Im Gegensatz dazu ist diese Auflage bei Zugriffen auf die Betriebssystem-Erweiterungen oder globale Code- und Datensegmente nicht erforderlich.

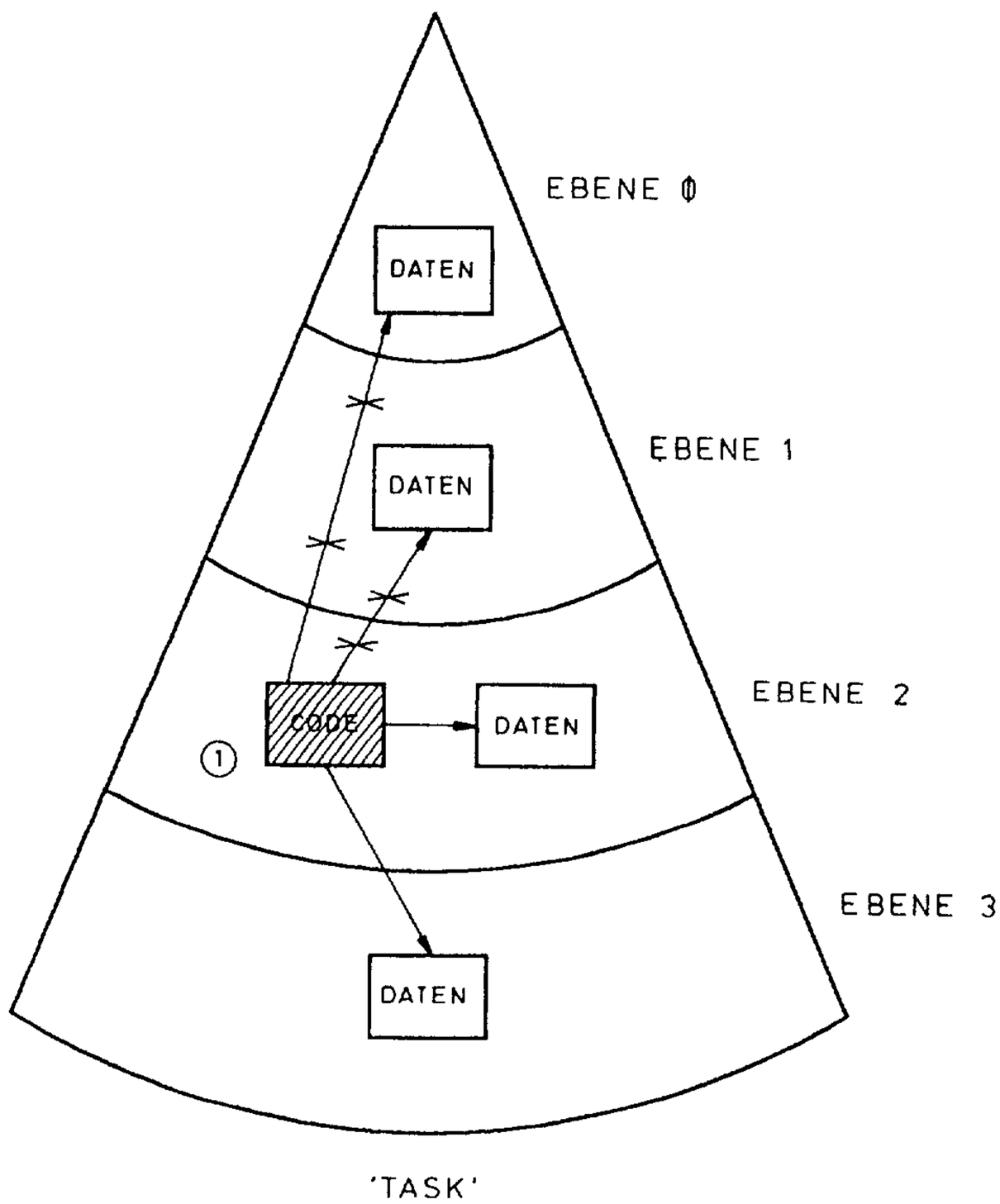
Damit innerhalb einer "Task" der erforderliche Schutz realisiert werden kann, stellt der 80286 eine Privileg-Hierarchie mit vier Ebenen zur Verfügung. Jedes Code- oder Datensegment hält sich auf einer dieser vier Privilegebenen auf. Dabei ist Ebene 0 die privilegierteste Ebene mit dem höchsten Vertrauensgrad, während Ebene 3 den niedrigsten Vertrauensgrad hat, d. h. am ehesten Übergriffen und Störmöglichkeiten anderer Prozeduren ausgesetzt ist.
Durch diese hierarchische Struktur können Schutzmechanismen implementiert werden, die anhand des folgenden Bildes erläutert werden sollen.



So benützt die 80286-Architektur Privilegebenen, um die Zugriffsfähigkeit eines Codes auf einen anderen Code oder auf Daten zu steuern.

- Da der Vertrauensgrad und damit die Zuverlässigkeit von Programmen immer geringer wird, je niedriger ihre Privilegebene ist, können solche Programme auf Datensegmente in höheren Privilegebenen ***nicht*** zugreifen.
- Allerdings ist der Zugriff eines Programms auf Datensegmente erlaubt, die sich entweder auf der gleichen Privilegebene wie das Programm oder auf einer niedrigeren Privilegebene aufhalten.
- Auf Codesegmente in einer höheren Privilegebene kann nur indirekt über einen programm-transparenten Steuermachanismus, ein sogenanntes "Gate", zugegriffen werden.
- Weiterhin ist es möglich, auf Codesegmente zuzugreifen, die mit dem aufrufenden Programm auf der gleichen Privilegebene liegen.
- Auf Codesegmente in einer niedrigeren Privilegebene kann ***nicht*** zugegriffen werden, weil ihr Vertrauensgrad geringer ist als der des aufrufenden Programms.

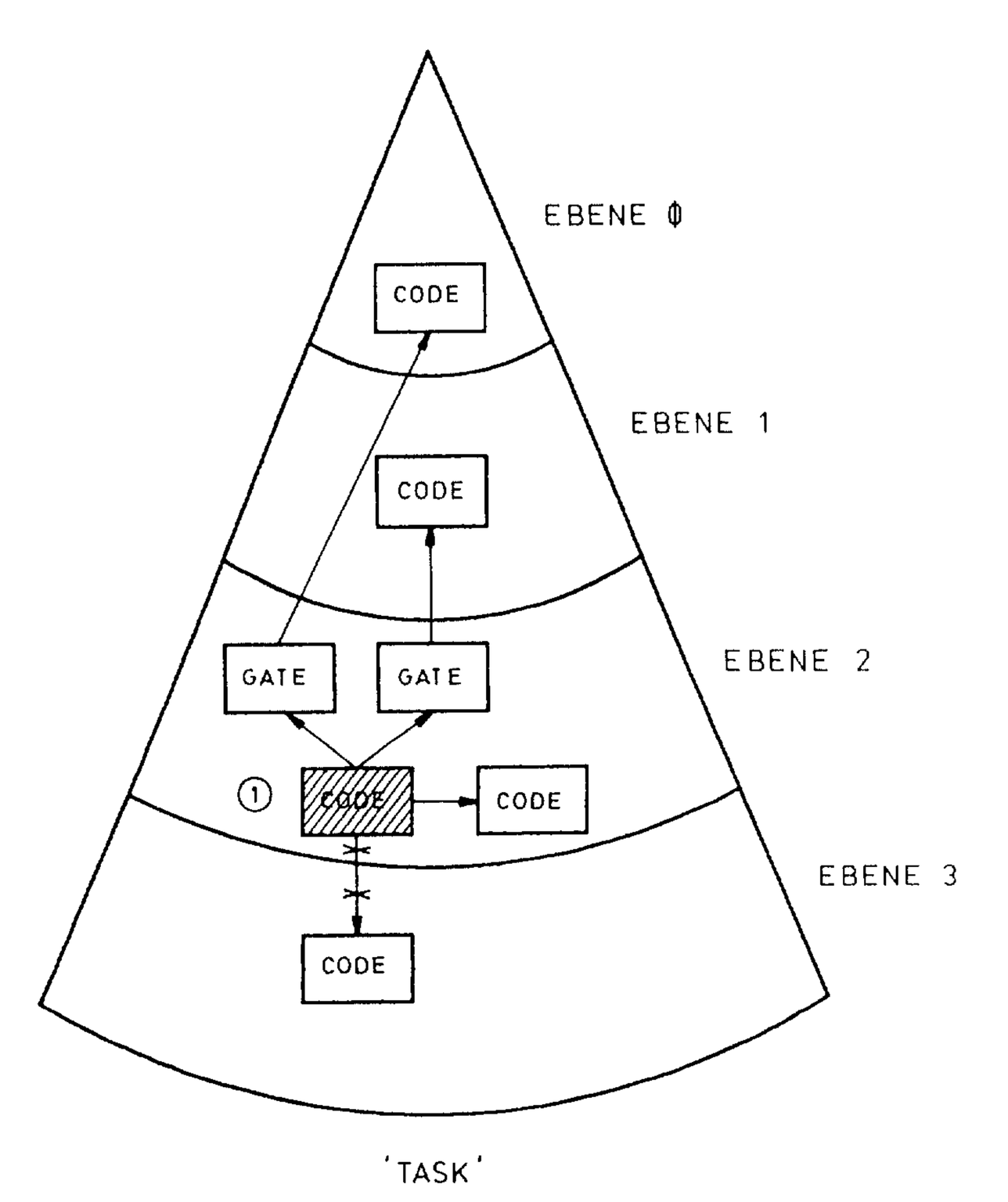
Wie der Vier-Ebenen-Mechanismus die Manipulation von Daten einschränkt, soll das folgende Beispiel verdeutlichen.



Es ist ersichtlich, daß die augenblicklich aktive "Task" einen Code in der Privilegebene 2 ausführt ①.
Auf die Daten in Ebene 2 und Ebene 3 kann der Code ohne Einschränkung zugreifen, während ein Zugriff auf die Daten in Ebene 1 und Ebene 0 ***nicht*** möglich ist.

Die Aussage dieses Beispiels kann auf den Schutz privater Betriebssystem-Daten übertragen werden.
Da sich die privaten Daten eines Betriebssystems typischerweise in der Privilegebene 0 befinden, sind diese vor Anwenderprogrammen, die sich im Normalfall in der Privilegebene 3 aufhalten, geschützt.

Wie der Vier-Ebenen-Mechanismus den unkontrollierten Zugriff auf Codesegmente verhindert, soll das folgende Beispiel verdeutlichen.



Es ist ersichtlich, daß die augenblicklich aktive "Task" einen Code in der Privilegebene 2 ausführt ①.
Dieser Code kann einen anderen Code in der gleichen Privilegebene (im Beispiel Ebene 2) aufrufen.

Da die Zuverlässigkeit des Codes in Ebene 3 geringer ist als die des Codes in Ebene 2, ist ein Zugriff auf den Code in Ebene 3 ***nicht*** möglich.

Damit ein Programm mit geringer Zuverlässigkeit nicht "wild" in höherprivilegierte Routinen springen kann, ist der direkte Zugriff auf diese Routinen ***nicht*** erlaubt.

So kann der Code in Ebene 2 die Codes in Ebene 1 und 0 nur über "Gates" erreichen, wobei diese "Gates" ***feste*** Einsprungadressen vorgeben. Dadurch ist ein unkontrollierter Zugriff ausgeschlossen.

Die Aussage dieses Beispiels kann auf den Schutz von privatem Betriebssystem-Code übertragen werden.
Da sich der private Code eines Betriebssystems typischerweise in der Privilegebene 0 befindet, ist dieser vor unkontrollierten Zugriffen der Anwenderprogramme, die sich im Normalfall in der Privilegebene 3 aufhalten, geschützt.

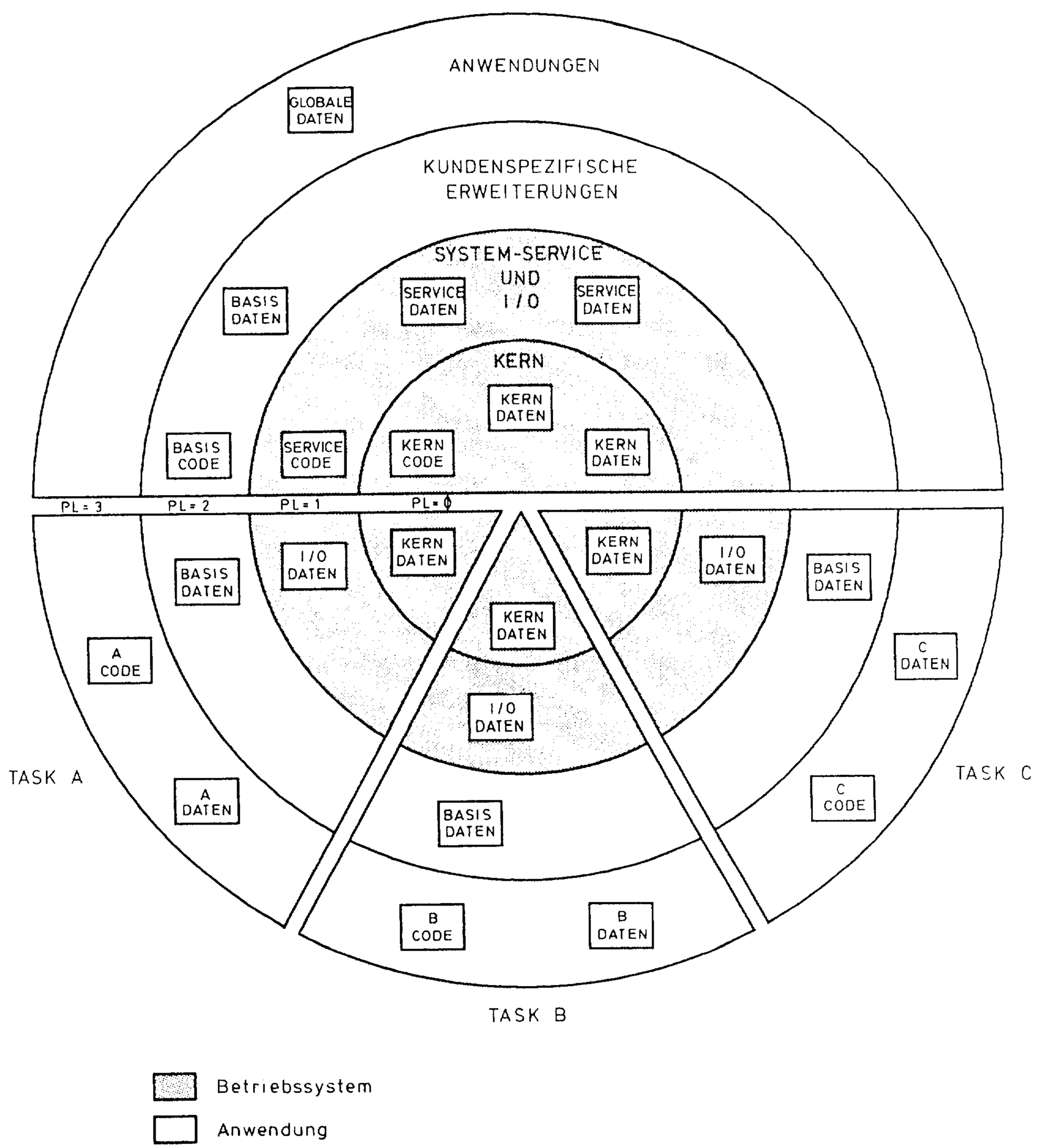
3.5 Struktur des Software-Systems

Die Privileg-Hierarchie der 80286-Architektur gibt dem System-Entwickler die Möglichkeit, Code und Daten auf die verschiedenen Privilegebenen und "Tasks" zu verteilen. Die daraus resultierende Software-Struktur beeinflußt direkt die Zuverlässigkeit, die Leistungsfähigkeit und die Anpassungsfähigkeit des Systems.

Wenn Software-Module auf verschiedene Privilegebenen und separate "Tasks" verteilt werden, ergeben sich hierfür einige Vorteile:

1) Es wird eine feinere "Körnung" des Schutzes erreicht.
2) Da jede "Task" nur eine bestimmte Aufgabe erledigen kann (z. B. Lesen einer Tastatur), erfolgt die Umschaltung auf andere "Tasks" automatisch. Dadurch ist die Synchronisation der anfordernden "Tasks" nicht notwendig.
3) Die Kommunikation zwischen Anwender-Software und Betriebssystem ist schneller.
4) Für alle "Tasks" ist es möglich, die gleichen Betriebssystem-Funktionen parallel auszuführen.

Das folgende Bild illustriert eine 80286-Software-Struktur, die alle vier Privilegebenen benützt, um ein Maximum an Vorteilen zu haben.

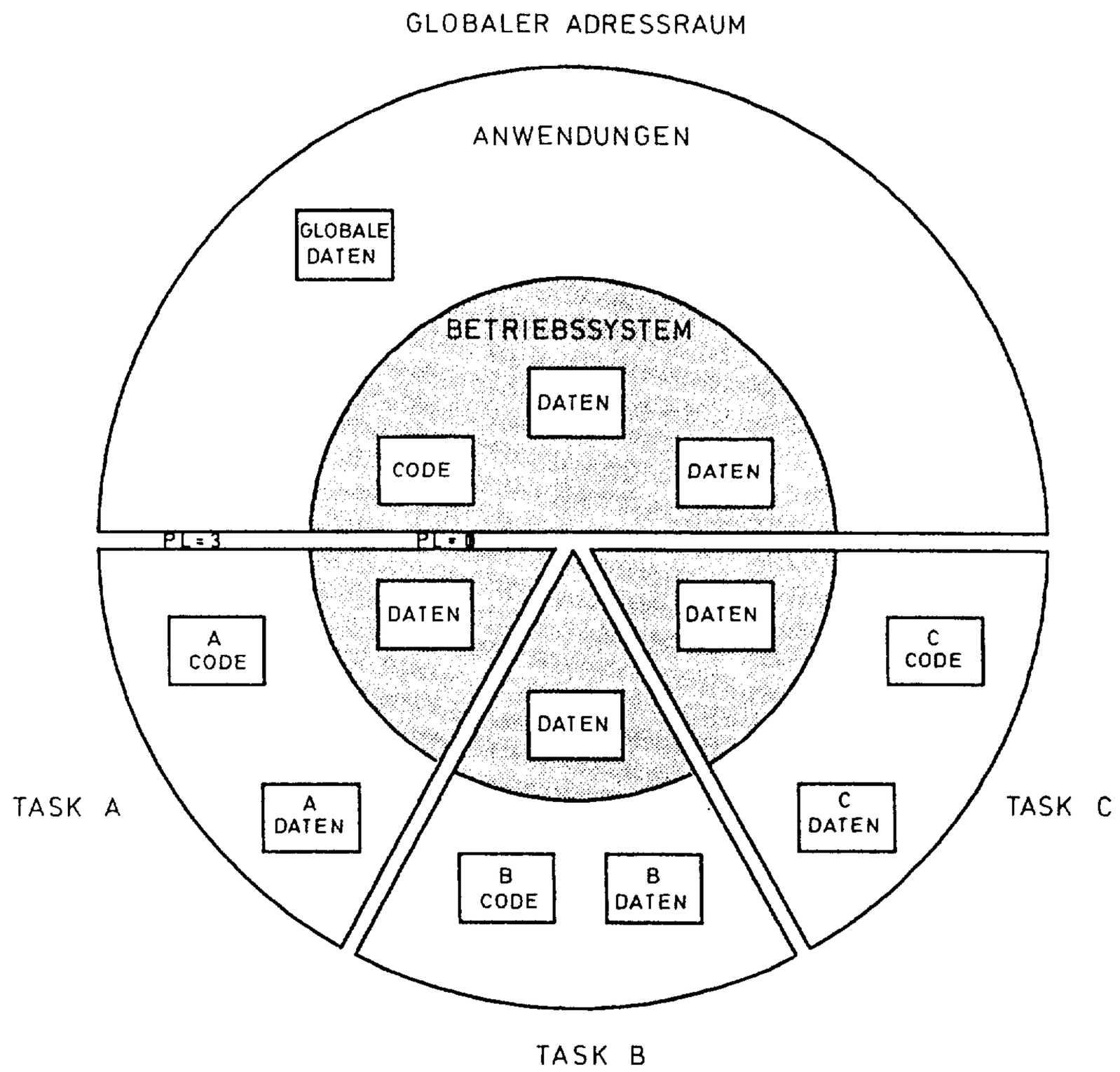


Die kritischen Prozeduren und Daten des Betriebssystem-Kerns sind vor allen anderen Prozeduren im System geschützt. Zu diesen kritischen Regionen gehören z. B. Deskriptor-Tabellen und Speicher-Zuweisungs-Prozeduren oder Informationen über "Tasks". Sie gehören daher zum Kern des Betriebssystems und liegen auf der Privilegebene 0 (PL = 0).

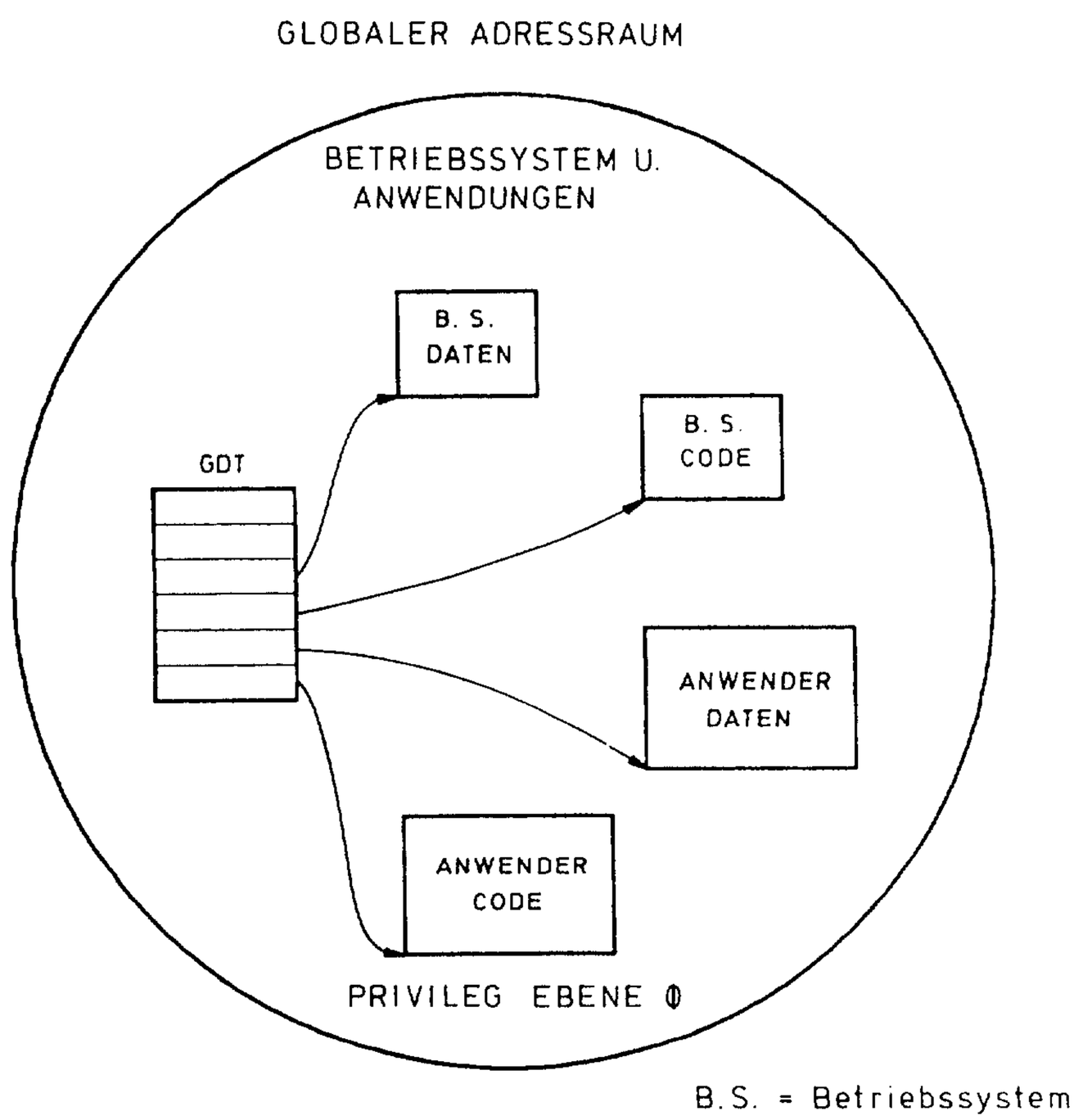
Jene Teile des Betriebssystems, die weniger zuverlässig sind, liegen auf der Privilegebene 1 (PL = 1) und sind deshalb noch vor Anwenderebenen geschützt. Die geringere Zuverlässigkeit dieser Betriebssystem-Teile ist entweder auf ihre natürliche Komplexität zurückzuführen, die z. B. für I/O-Subsysteme charakteristisch ist, oder auf gelegentlich durch sie ausgelöste Maßnahmen, die den Datendurchsatz erhöhen sollen.

Die Anwender-Logik hat zwei Ebenen, sodaß kundenspezifische Betriebssystem-Erweiterungen wie z. B. das Management von Basisdaten oder die Verwaltung von Dateien vom Zugriff des reinen Anwender-Codes geschützt werden können.

Wie das folgende Bild zeigt, ist es ***nicht*** unbedingt notwendig, alle vier Privilegebenen zu benützen. So kann der Anwender die angegebenen zwei Privilegebenen bevorzugen, wenn er sein System an die traditionellen Multi-Tasking-Systeme angleichen will. Solche Systeme bieten nur zwischen zwei Ebenen Schutz, wobei die eine Ebene duch die Anwendungen und die andere Ebene durch das Betriebssystem definiert ist.

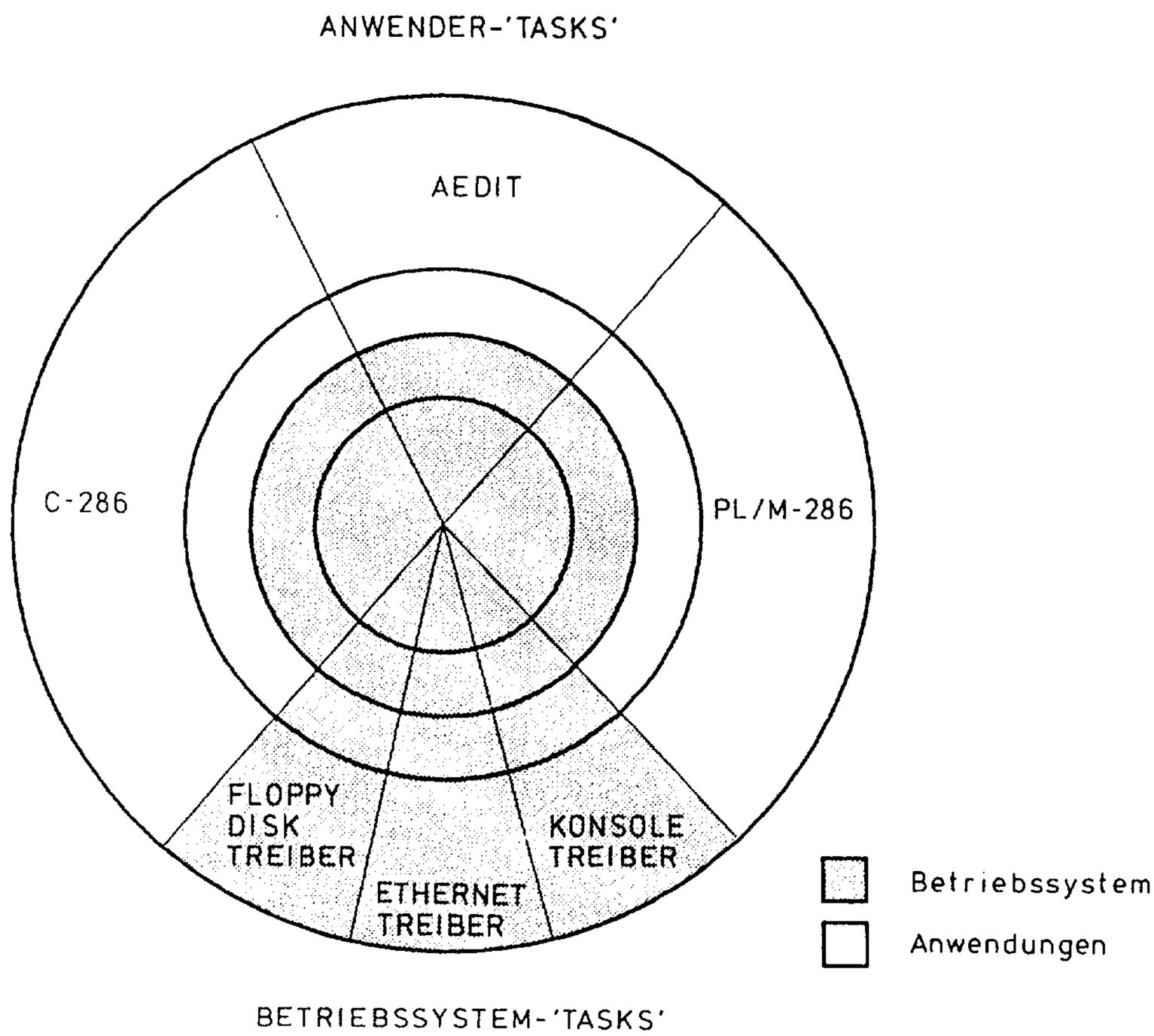


Wie das folgende Bild illustriert, kann der 80286 auch ein Ein-Ebenen-System emulieren.
Dieses System kann in der Initialisierungs-Phase nützlich sein, wenn der Wechsel von einer ungeschützten Umgebung in eine geschützte erfolgt. Allerdings nimmt ein solches System die vielen Vorteile des Schutzes, die der 80286 bietet, nicht in Anspruch. So ist beispielsweise das Betriebssystem vor Anwender-Software ***nicht*** geschützt.



Manchmal ist es besser, einige Betriebs-System-Funktionen als unabhängige Betriebs-System-"Tasks" und nicht als privilegierte Prozeduren innerhalb einer "Task" zu strukturieren. Dafür eignen sich bestimmte I/O-Funktionen. Da diese Funktionen erst durch I/O-Bausteine aktiviert werden, ist es vorteilhaft, sie als separate "Tasks" zu behandeln, die völlig asynchron ablaufen können.

Damit die Zuverlässigkeit des Systems durch die Komplexität der I/O-Routinen nicht beeinträchtigt wird, ist ihr gegenseitiger Schutz durch die "Task"-Grenzen gegeben. Das nachstehende Bild illustriert ein solches System.



3.6 Daten-Segment/Executable-Segment-Deskriptoren

Jeder Zugriff eines Segments auf ein anderes Segment ist nur indirekt über einen Deskriptor möglich. Er enthält alle Informationen über das Segment, auf das der Zugriff erfolgen soll. Jedes Segment muß mindestens einen Deskriptor haben. Im anderen Fall würde das bedeuten, daß das Segment im Speicherraum nicht vorhanden ist.

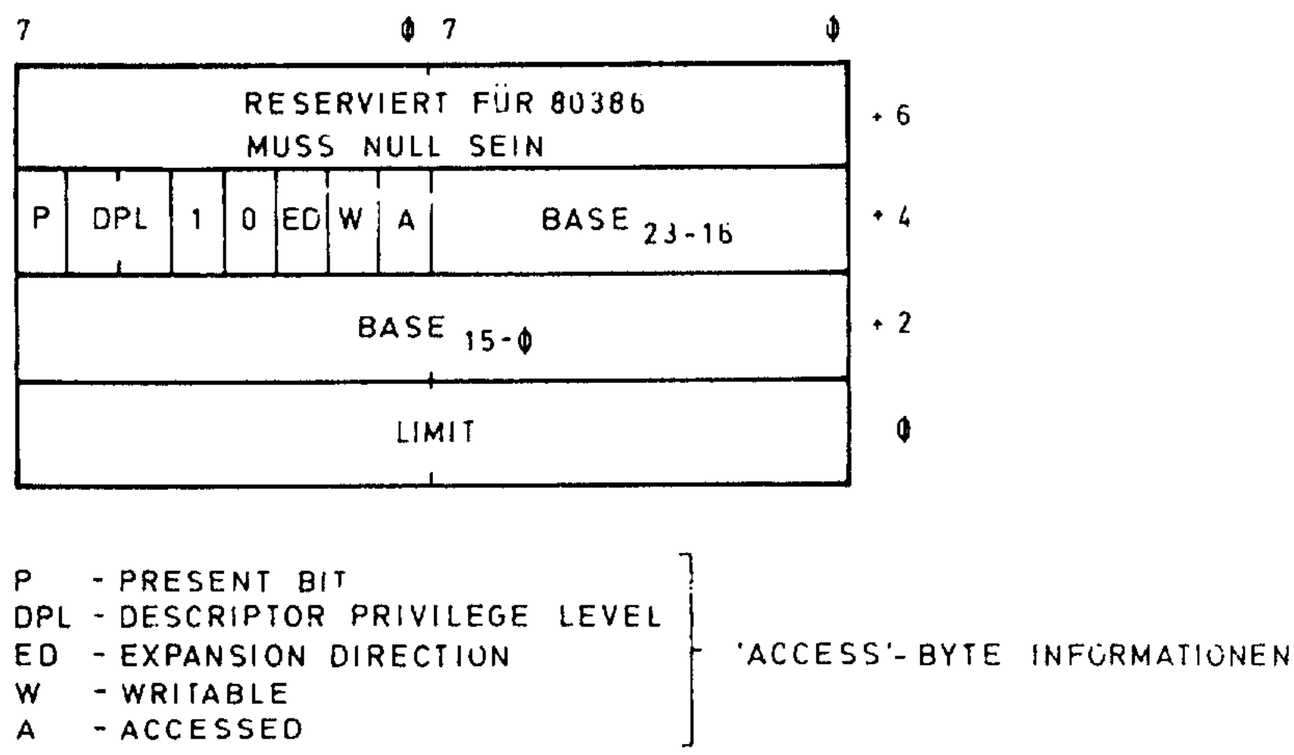
Die Deskriptoren werden entweder vom "System Builder" BLD286 für statische Systeme oder von Betriebssystem-Prozeduren für dynamische Systeme festgelegt. Dabei sind ihre Existenz und Funktion für den Anwender-Programmierer völlig "unsichtbar".

Die in einem 80286-System bekannten Deskriptor-Formate lassen sich in vier Klassen einteilen, wobei zunächst die Klasse der

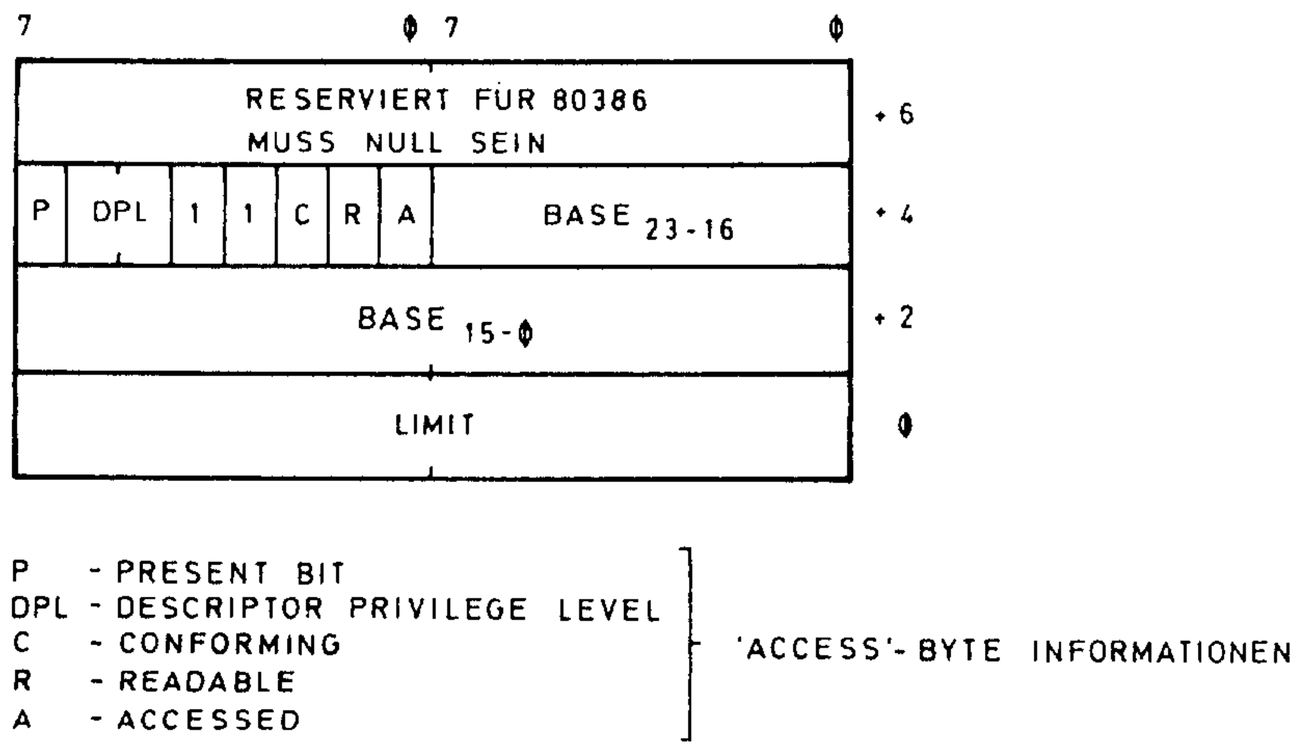
- Daten-Segment-Deskriptoren und der
- Executable-Segment-Deskriptoren

betrachtet werden sollen.

Daten-Segment-Deskriptoren beziehen sich ausschließlich auf solche Segmente, die entweder System- oder Anwenderdaten enthalten. Dazu gehören ***auch*** Stack-Segmente.
Das folgende Bild zeigt das Format eines Daten-Segment-Deskriptors.



Executable-Segment-Deskriptoren beziehen sich ausschließlich auf Segmente, die ausführbare Befehle enthalten.
Das folgende Bild zeigt das Format eines Executable-Segment-Deskriptors.



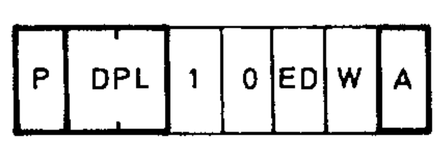
Es ist zu erkennen, daß in beiden Deskriptor-Formaten einige gemeinsame Felder existieren. Diese sind

- das Limit-Feld (Segment-Obergrenze)
- das Base-Feld (Segment-Basisadresse) und die
- Bit-Felder P (Present), DPL (Descriptor Privilege Level) und A (Accessed) im "Access"-Byte.

Da die Limit- und Base-Felder bereits früher besprochen worden sind, sollen zunächst die gemeinsamen Bit-Informationen im "Access"-Byte der beiden Deskriptoren diskutiert werden.

> **ZUR ERINNERUNG:**
> Das "Access"-Byte legt die Zugriffsrechte eines Segments auf ein anderes Segment fest.

3.6.1 Gemeinsamkeiten der "Access-Bytes"



'ACCESS-BYTE' DES DATEN-SEGMENT-DESKRIPTORS

'ACCESS-BYTE' DES EXECUTABLE-SEGMENT-DESKRIPTORS

Present-Bit (P)
Dieses Bit ist gesetzt (P = 1), wenn das durch einen Deskriptor adressierte Segment im Speicher gegenwärtig ist. Im anderen Fall ist dieses Bit gelöscht (P = 0). Jeder Zugriff auf ein Segment, dessen Present-Bit gelöscht ist, stellt eine Ausnahme-Situation dar, die allerdings einem Betriebssystem für dynamische Anwendungen die Gelegenheit gibt, das Segment vom virtuellen Speicher zu laden.

Accessed-Bit (A)
Der Prozessor setzt dieses Bit (A = 1) immer dann, wenn ein Deskriptor in ein Segment-Register geladen wird. Damit A manipuliert werden kann, muß sich der Deskriptor im RAM-Bereich aufhalten, da ein EPROM-Deskriptor-Feld ***nicht*** beschreibbar ist. Das Betriebssystem hat die Möglichkeit, dieses Bit periodisch zu testen und zu löschen und kann dadurch die Frequenz des Segmentzugriffs verfolgen.
Dieses Bit zeigt auch an, ob ein Segment in den Hintergrund-Speicher ausgelagert werden muß, bevor der vom Segment besetzte RAM-Bereich wiederbenützt wird.

Descriptor Privilege Level (DPL)
Der Wert in diesem Feld definiert die Privilegebene des durch den Deskriptor adressierten Segments.

TABELLE DER INITIALISIERUNGSWERTE

DPL		SEGMENT- PRIVILEG-EBENE
0	0	0
0	1	1
1	0	2
1	1	3

Das DPL-Feld kann für ein statisches System oder für den residenten Teil eines dynamischen Systems vom "System Builder" BLD286 initialisiert werden.
Wenn Segmente dynamisch geladen werden, übernehmen Prozeduren des Betriebssystems die Initialisierung von DPL.

3.7 Das "Access Byte" des Executable-Segment-Deskriptors

3.7.1 Das Readable-Bit (R)

Wenn dieses Bit gelöscht ist (R = 0), können Prozeduren in anderen Segmenten dieses Code-Segment ***nicht*** lesen. Dies bedeutet, daß der Inhalt dieses Segments nur ausgeführt werden kann und als "Execute Only"-Segment (Typ EO) spezifiziert ist.

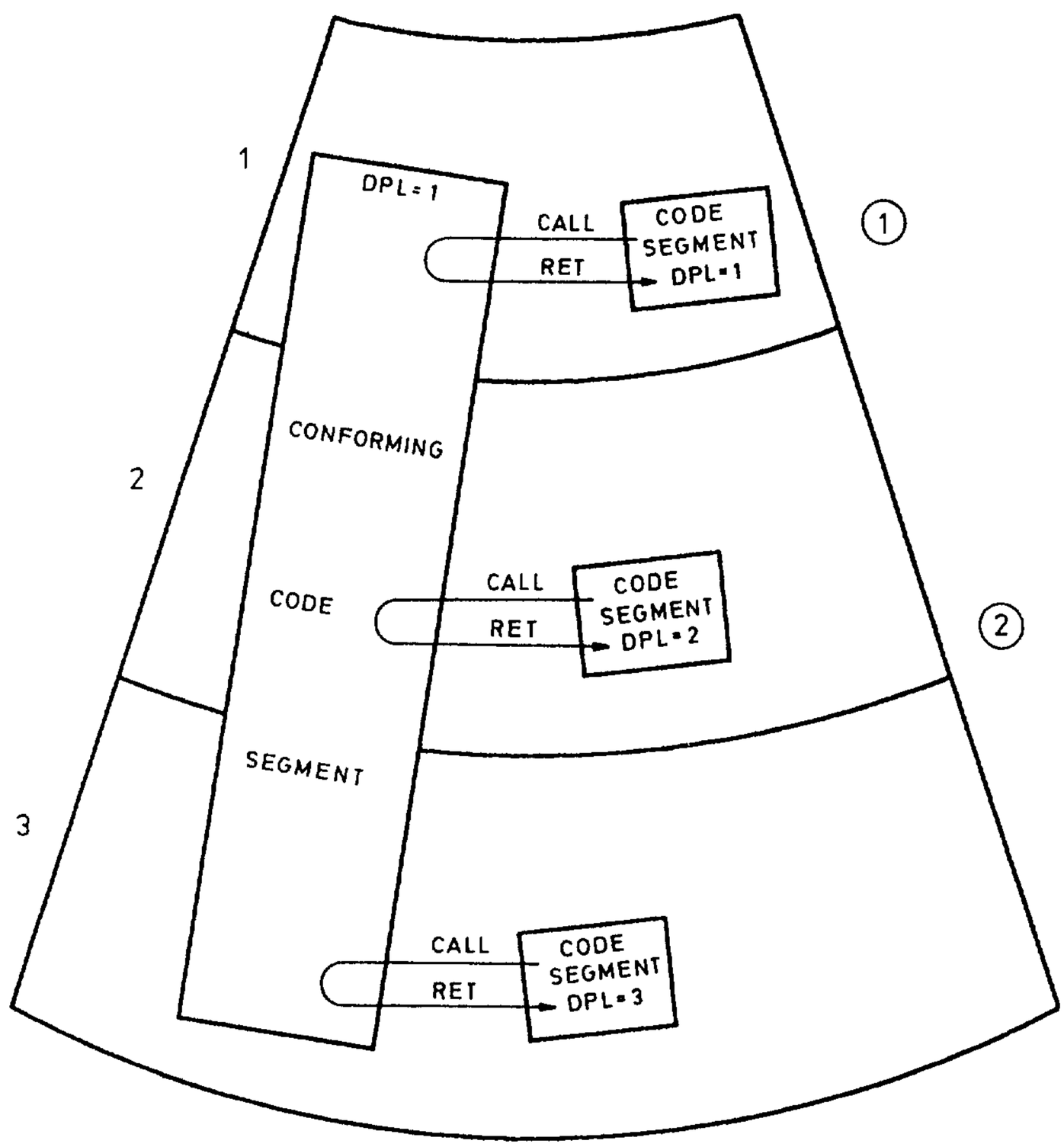
Sollte ein ausführbares Segment auch lesbare Daten (z. B. Konstanten) enthalten, muß das R-Bit gesetzt sein (R = 1). In diesem Fall ist das Segment als "Execute- and Readable"-Segment (Typ ER) spezifiziert.

3.7.2 Das Conforming-Bit (C)

Dieses Bit wird nur für "Execute Only"-Segment-Deskriptoren benützt. Ist das "Conforming"-Bit des aufgerufenen Segments gelöscht (C = 0), liegt die einfachere Situation vor. Die aufgerufene Prozedur hält sich in genau dem Segment auf, dessen Privilegebene durch das DPL-Feld des zugehörigen Deskriptors definiert ist.

Ist das "Conforming"-Bit des aufgerufenen Segments gesetzt (C = 1), wird in diesem Fall die aufgerufene Prozedur auf der gleichen Privilegebene wie die aufrufende Prozedur ausgeführt.

Die folgende Graphik soll die Arbeitsweise eines "Conforming"-Segments illustrieren.

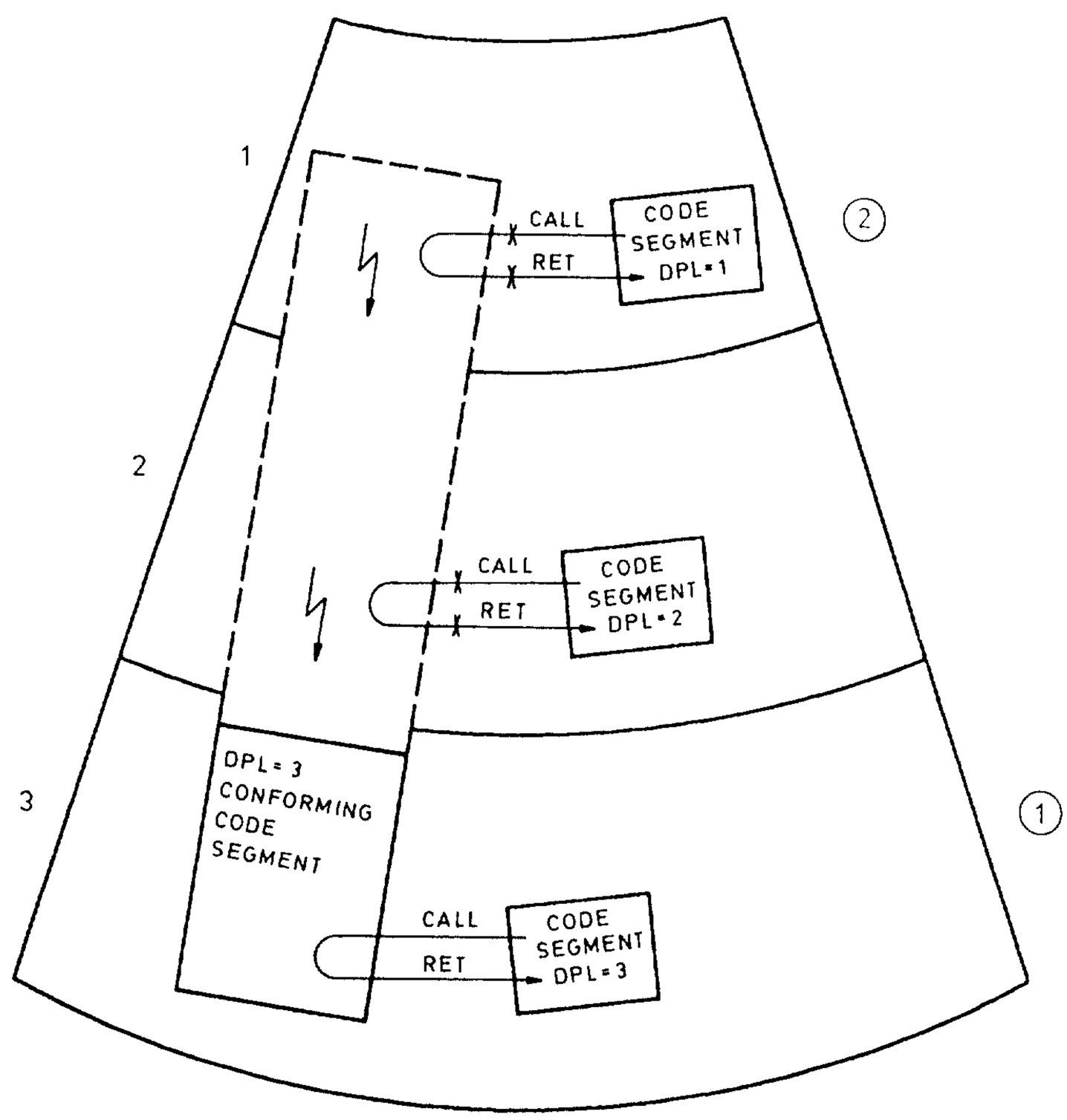


Es ist ersichtlich, daß sich das "Conforming Code"-Segment auf der Privilegebene 1 (DPL = 1) befindet. Die Privilegebene des "Conforming"-Segments ändert sich nicht, wenn sich das aufrufende Code-Segment auf der gleichen Ebene befindet (im Beispiel DPL = 1) ①.

Sind aber die Privilegebenen der aufrufenden Code-Segmente numerisch ***größer*** (im Beispiel DPL = 2 und DPL = 3) als die Ebene des aufgerufenen "Conforming"-Segments, wird das "Conforming"-Segment auf diesen Ebenen ausgeführt ②.

Die Leistungsmerkmale eines "Conforming"-Segments können nicht ausgenützt werden, wenn die Privilegebene des aufrufenden Code-Segments numerisch ***kleiner*** ist als die Ebene des aufzurufenden "Conforming"-Segments.

Das folgende Bild soll dies verdeutlichen.



Es ist ersichtlich, daß sich das "Conforming Code"-Segment auf der Privilegebene 3 (DPL = 3) befindet. Seine Privilegebene bleibt erhalten, wenn es von einem Code-Segment auf der gleichen Ebene aufgerufen wird (im Beispiel DPL = 3) ①.

Der Versuch eines Zugriffs auf das "Conforming"-Segment durch die Code-Segmente in den Privilegebenen 1 und 2 stellt eine Ausnahme-Situation dar und ist daher nicht erlaubt ②.

Die Leistungsmerkmale der "Conforming"-Segmente sind immer dann nützlich, wenn bestimmte Prozeduren (z. B. mathematische Routinen) für andere Prozeduren, die auf niedrigeren Privilegebenen laufen, ***ohne*** Privilegwechsel verfügbar sein sollen.

3.8 Das "Access Byte" des Daten-Segment-Deskriptors

3.8.1 Das Writable-Bit (W)

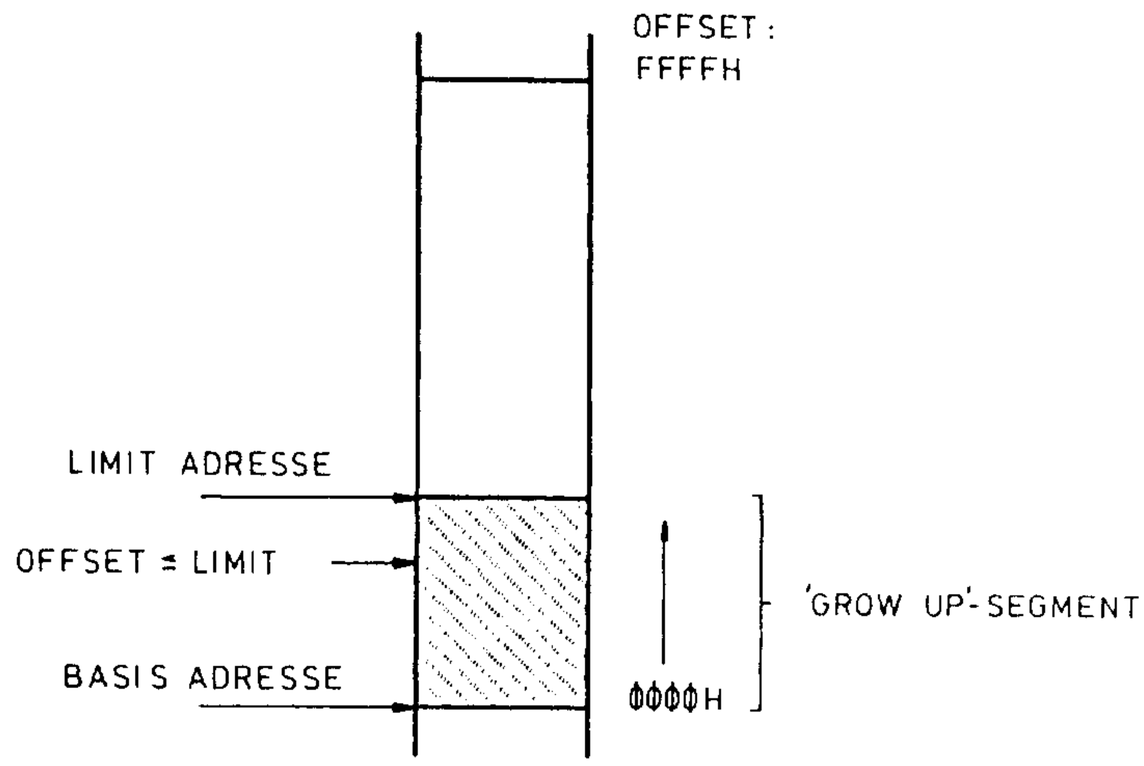
Wenn dieses Bit gesetzt ist (W = 1), hat die CPU die Erlaubnis, in das Segment zu schreiben. Soll das Segment vor einem Schreibversuch geschützt werden, muß dieses Bit gelöscht sein (W = 0). In diesem Fall handelt es sich um ein "Read Only"-Segment (Typ RO).

So sind Übersetzungs-Tabellen ein Beispiel für Daten, die durch ihren besonderen Anspruch auf Schutz in "Read Only"-Segmenten gespeichert werden.

3.8.2 Das Expansion-Direction-Bit (ED)

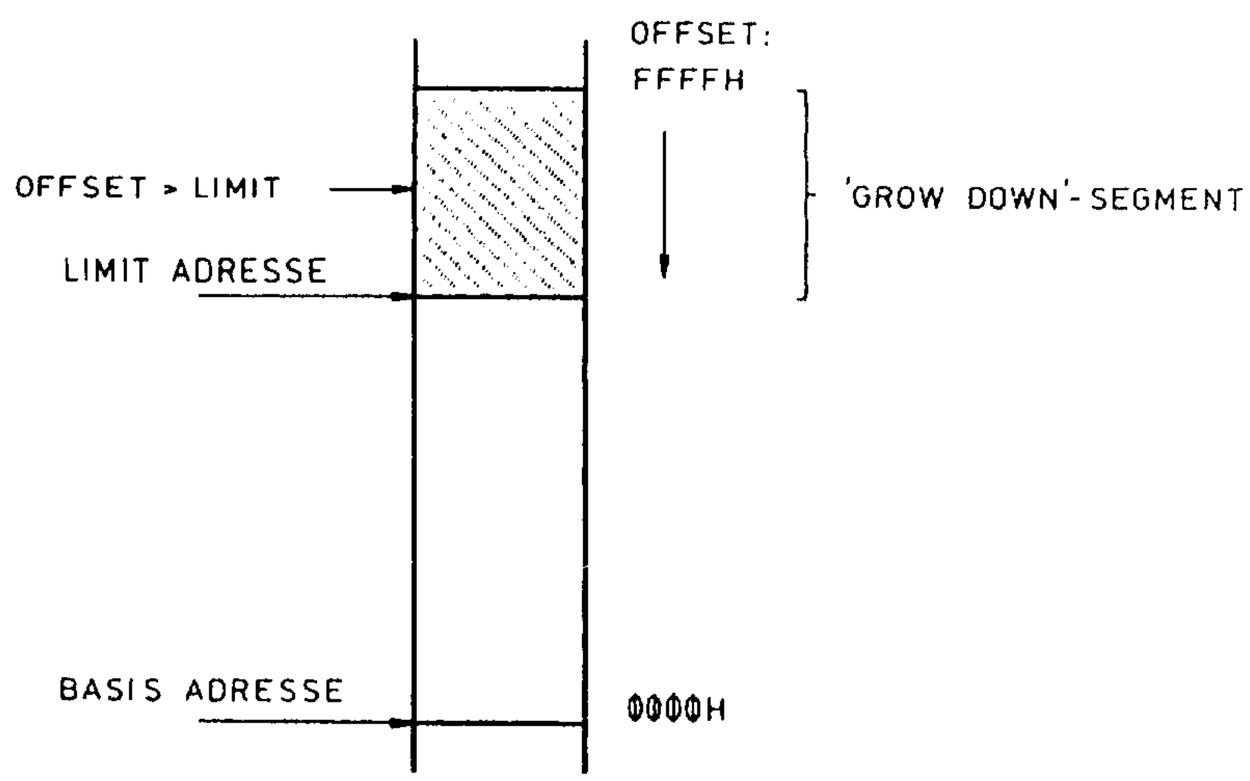
Daten-Segmente können sowohl Stack-Daten als auch andere Datenstrukturen enthalten.
Ein "normales" Daten-Segment wird durch Offsetwerte adressiert, die innerhalb des Bereichs von 0 und der Größe des Segments liegen. Ist das ED-Bit gelöscht (ED = 0), sind Offsetwerte zwischen 0000H bis einschließlich dem Segment-Obergrenzwert (Limit) erlaubt. Hat das Limit-Feld im Deskriptor den Wert FFFFH, enthält das Daten-Segment 65536 Bytes.

Der Wert 0 im ED-Feld gibt an, daß das Daten-Segment, beginnend bei seiner Basisadresse, in Richtung Segment-Obergrenze wächst. Daher wird ein solches Segment als ***"Grow Up"***-Segment bezeichnet. Das folgende Bild soll dies verdeutlichen.



Während sich die meisten Datenstrukturen in Richtung höherer Adressen ausbreiten, expandieren Stack-Daten in Richtung niedrigerer Adressen.
Damit das Verhalten des Stacks im Daten-Segment berücksichtigt werden kann, muß das ED-Bit gesetzt sein. Ein mit ED = 1 gekennzeichnetes Daten-Segment wird als ***"Grow Down"***-Segment bezeichnet.

Alle Offsets, die auf ein solches Segment Bezug nehmen, müssen eindeutig ***größer*** sein als die Segmentobergrenze.
Im Gegensatz dazu müssen die Offsets von "Grow Up"-Segmenten ***kleiner*** oder ***gleich*** der Segment-Obergrenze sein.
Das folgende Bild zeigt ein "Grow Down"-Segment.



Hat das Limit-Feld im Deskriptor den Wert 0, enthält das "Grow Down"-Segment $2^{16} - 1$ (65535) Bytes.
Mit einem Limit-Wert von FFFFH hat das "Grow Down"-Segment eine Größe von 0 Bytes.

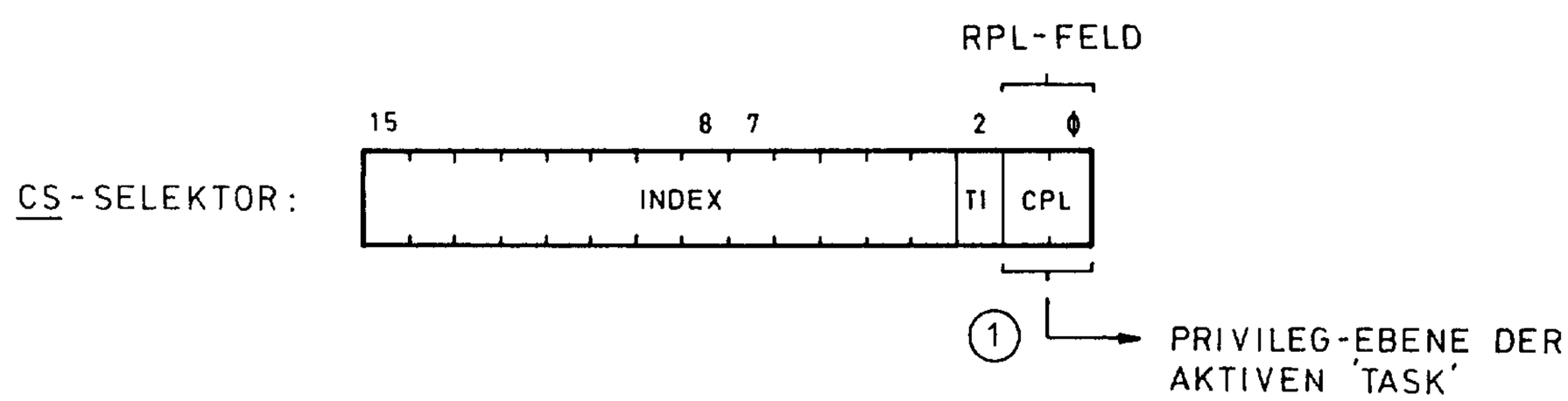
Da die Offset-Werte von "Grow Down"-Segmenten im Bereich von FFFFH bis Limit + 1 liegen, kann man die volle Größe des Segments nur bekommen (2^{16} Bytes), wenn man es als "Grow Up"-Segment deklariert.

Sowohl das "Grow Up"-Segment als auch das "Grow Down"-Segment können als Stack benützt werden.

3.9 Ablaufsteuerung eines Programms ohne Privileg-Wechsel

3.9.1 Privilegebene einer "Task"

Jede "Task" ist zu einem bestimmten Zeitpunkt nur auf ***einer*** Privilegebene aktiv und führt ein Code-Segment dieser Ebene aus.
Das Privileg-Attribut einer "Task" ist im RPL-Feld (Requestor's Privilege Level) des ***CS***-Selektors gespeichert und wird als "Current Privilege Level" (CPL) bezeichnet. CPL spezifiziert die Privilegebene der aktiven "Task" ①.



Der Zugriff auf ein Code-Segment erfolgt immer dann, wenn das CS-Register mit einem Selektor geladen wird. Innerhalb eines ausgewählten Segments kann der Ablauf des Programms durch einen der folgenden Befehle gesteuert werden:

- JMP Offset
- CALL Offset
- RET; (der Offset wird dem Stack entnommen)

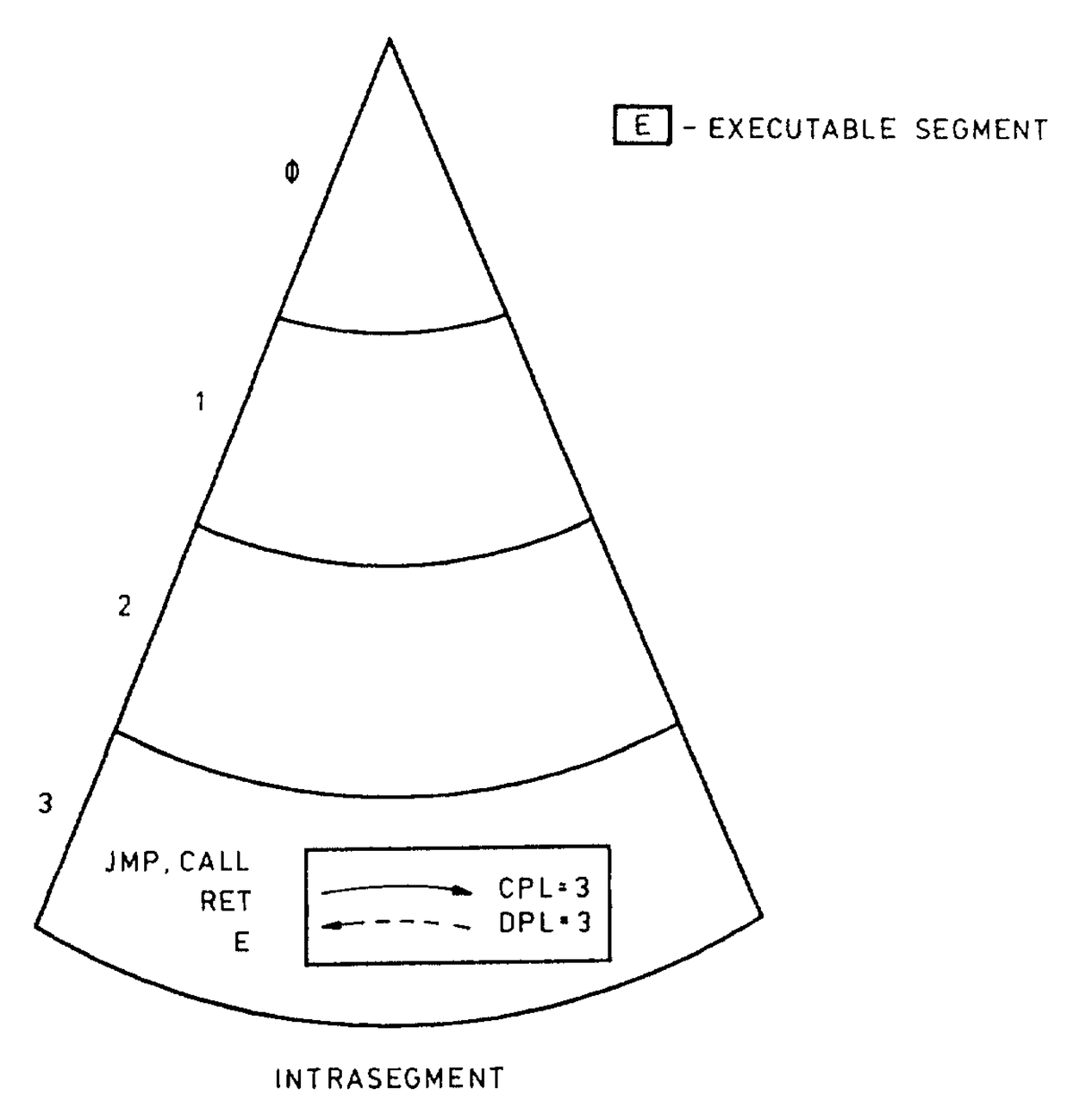
Da diese Befehle nur die Offset-Komponente ("Instruction Pointer" IP) des virtuellen Adresszeigers verändern, werden sie als Intra-Segment-Steuerbefehle bezeichnet.

Ist es notwendig, daß während der Ausführung eines Programms von einem Code-Segment auf ein anderes gewechselt werden muß, ist dies durch einen der folgenden Befehle realisierbar:

- JMP Selektor: Offset
- CALL Selektor: Offset
- RET; (der Selektor und der Offset werden dem Stack entnommen)

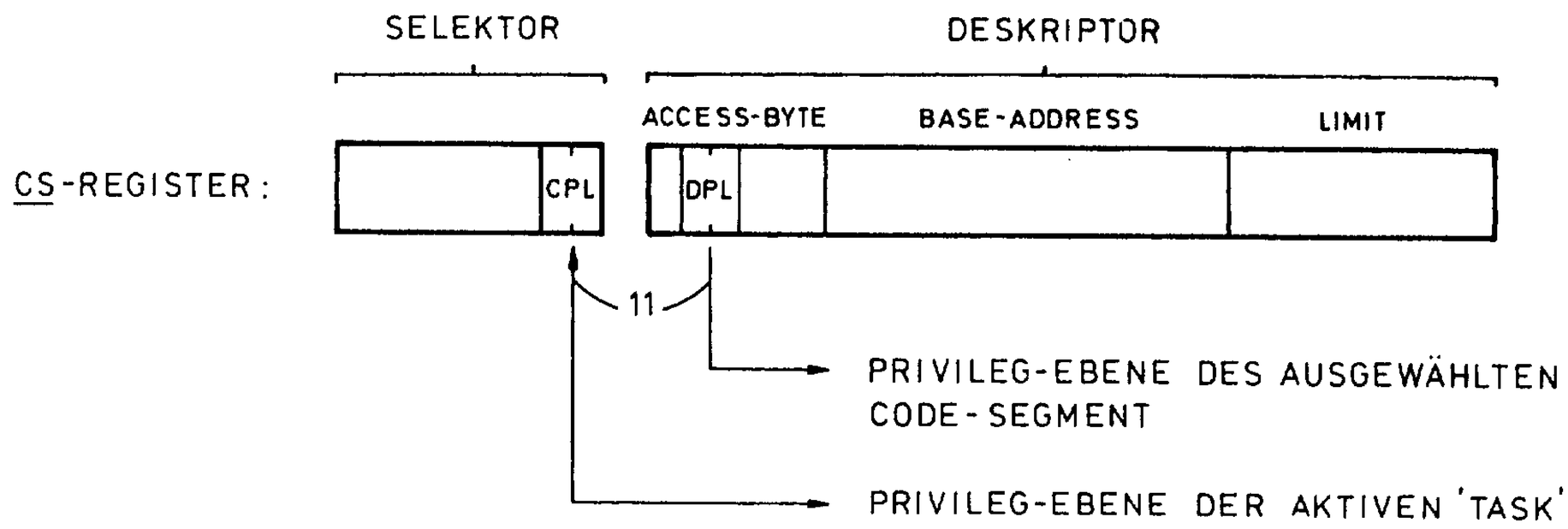
Da diese Befehle sowohl die Selektor-Komponente (Register CS) als auch die Offset-Komponente ("Instruction Pointer" IP) des virtuellen Adresszeigers verändern, werden sie als Inter-Segment-Steuerbefehle bezeichnet.

Das folgende Bild illustriert den ***Intra***-Segment-Ablauf eines Programms auf der Privilegebene 3.

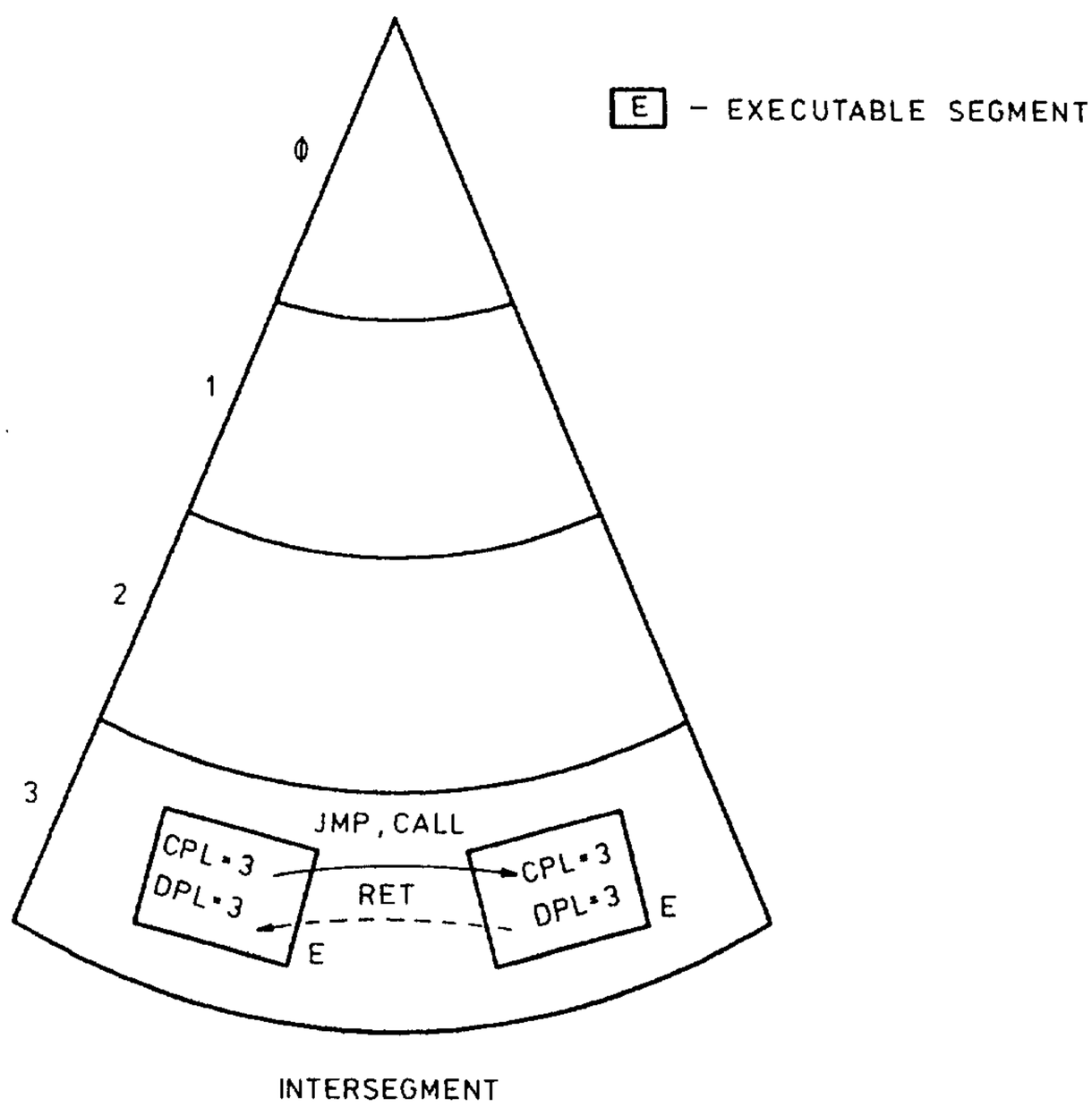


Der Selektor wählt den Deskriptor eines ausführbaren Segments aus. Da die Privilegebene (DPL) im Deskriptor dieses Segments den Wert 3 hat, wird das CPL-Feld des CS-Selektors mit diesem Wert initialisiert. Dadurch spezifiziert CPL die Privilegebene (3) der aktiven "Task".

Das folgende Bild verdeutlicht diesen Mechanismus:



Das folgende Bild illustriert den ***Inter***-Segment-Ablauf eines Programms auf der Privilegebene 3.



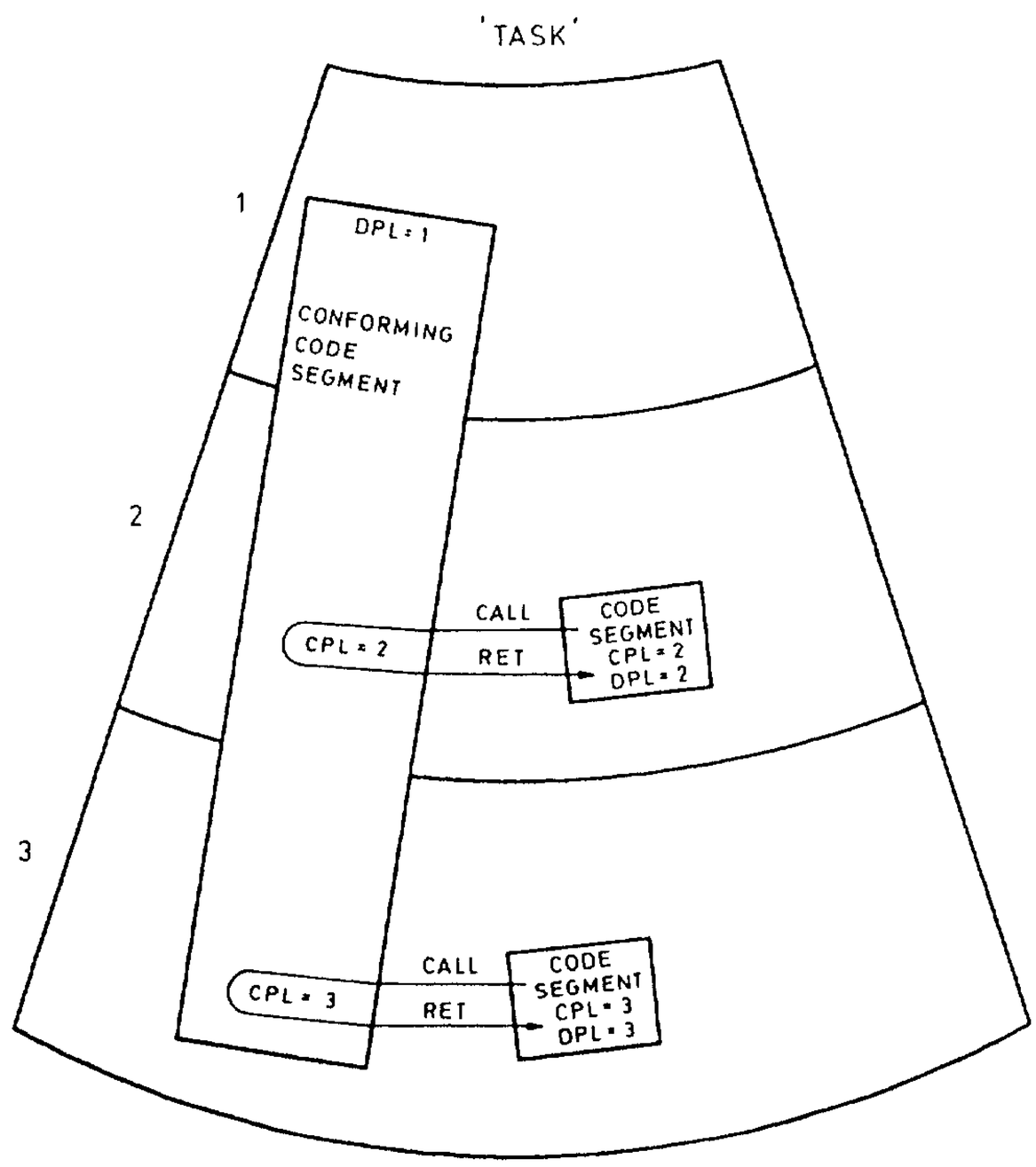
Da der Segment-Wechsel auf der ***gleichen*** Privilegebene erfolgt, wird ein solches Programm als "intralevel"-gesteuert bezeichnet. Der Segment-Wechsel wird eingeleitet, indem der CS-Selektor neu initialisiert wird. Da die Privilegebene (DPL) des Zielsegments mit der des aufrufenden Segments identisch ist, ändert sich das CPL-Feld im neuen CS-Selektor nicht.
Die Privilegebene der aktiven "Task" bleibt daher erhalten.

Jede "Task" kann Segmente auf der ersten, zweiten, dritten oder vierten Ebene enthalten, die alle zu passenden Zeiten benützt werden können. Da eine "Task" jeweils nur auf ***einer*** Privilegebene aktiv ist, hat der Übergang auf ein Code-Segment in einer anderen Ebene zur Folge, daß sich die Privilegebene der "Task" ändert. Solche Übergänge unterliegen strengen Regeln, die sorgfältig eingehalten werden müssen.

Eine "normale" Privileg-Anforderung gehorcht diesen Regeln automatisch, da die augenblickliche Privilegebene (CPL) des anfordernden JMP- oder CALL-Befehls mit der Privilegebene (DPL) des aufgerufenen Segments identisch ist (DPL = CPL).

Wie bereit früher erwähnt, können Code-Segmente in privilegierteren Ebenen nur über sogenannte "Gates" erreicht werden. Eine Ausnahme stellen allerdings "Conforming"-Code-Segmente dar. Hier darf die Privilegebene des angeforderten "Conforming"-Segments (DPL) numerisch kleiner sein als CPL des aufrufenden Segments (DPL $\leqslant$ CPL).
Die "Task" führt das "Conforming"-Segment auf den jeweiligen Privilegebenen der aufrufenden Code-Segmente aus.

Das folgende Bild soll diese Situationen verdeutlichen:



Es ist ersichtlich, daß sich das "Conforming"-Segment auf der Privilegebene 1 (DPL = 1) befindet, die "Task" aber dieses Segment auf den Ebenen 2 (CPL = 2) oder 3 (CPL = 3) ausführt.

3.9.2 **Programmierbeispiel:** Intra-Level-Steuerung

Das folgende Programm-Fragment zeigt, wie ohne Privileg-Wechsel (Intra-Level) ein Programmablauf entweder ***intra***segment oder ***inter***segment gesteuert werden kann.

```
iAPX286 MACRO ASSEMBLER    B9


PC 16-20 - iAPX286 MACRO ASSEMBLER VX.X ASSEMBLY OF MODULE B9
OBJECT MODULE PLACED IN B9.OBJ
ASSEMBLER INVOKED BY:  ASM286.86 B9.SRC


LOC  OBJ                  LINE     SOURCE

                            1             NAME    B9
                            2
----                        3     CODE    SEGMENT EO                    ;Nur ausfuehrbares Segment mit
                            4                                           ;Basis-Adresse=1000H und DPL=3
                            5                                           ;(Festlegung durch BLD286)
                            6             ;
                            7             ;                             ;Normale Befehlssequenz
                            8             ;
0000 E80000                 9             CALL    INTRA                 ;Prozeduraufruf intrasegment-direkt
                           10             ;
                           11             ;                             ;Normale Befehlssequenz
0003                       12     INTRA   PROC   NEAR ;*****************************************************
                           13             ;
                           14             ;                             ;Prozedurkoerper
                           15             ;
0003 C3                    16             RET
                           17     INTRA   ENDP        ;*****************************************************
                           18             ;
                           19             ;                             ;Normale Befehlssequenz
                           20             ;
0004 9A0000----      R     21             CALL FAR PTR INTER            ;Prozeduraufruf intersegment-direkt
                           22             ;
                           23             ;                             ;Normale Befehlssequenz
----                       24     CODE    ENDS
                           25
                           26
----                       27     CODE1   SEGMENT EO                    ;Nur ausfuehrbares Segment mit
                           28                                           ;Basisadresse=2000H und DPL=3
                           29                                           ;(Festlegung durch BLD286)
                           30             ;
                           31             ;                             ;Normale Befehlssequenz
                           32             ;
0000                       33     INTER   PROC   FAR  ;*****************************************************
                           34             ;
```

```
                    35          ;                                   ;Prozedurkoerper
                    36          ;
0000 CB             37          RET
                    38   INTER  ENDP        ;*****************************************************
                    39          ;
                    40          ;                                   ;Normale Befehlssequenz
----                41   CODE1  ENDS
                    42
                    43          END

ASSEMBLY COMPLETE,   NO WARNINGS,   NO ERRORS
```

Das Programm enthält die beiden ausführbaren Code-Segmente (Typ EO) CODE und CODE1. Für beide Segmente generiert BLD286 je einen Deskriptor. Der eine Deskriptor beschreibt das CODE-Segment mit der Basisadresse 1000H und der Privilegebene 3 (DPL = 3). Der andere Deskriptor beschreibt das CODE1-Segment mit der Basisadresse 2000H und ebenfalls der Privilegebene 3 (DPL = 3).

Beide Segmente enthalten Prozeduren unterschiedlichen Typs. Im CODE-Segment ist die NEAR-Prozedur INTRA, und im CODE1-Segment ist die FAR-Prozedur INTER enthalten.

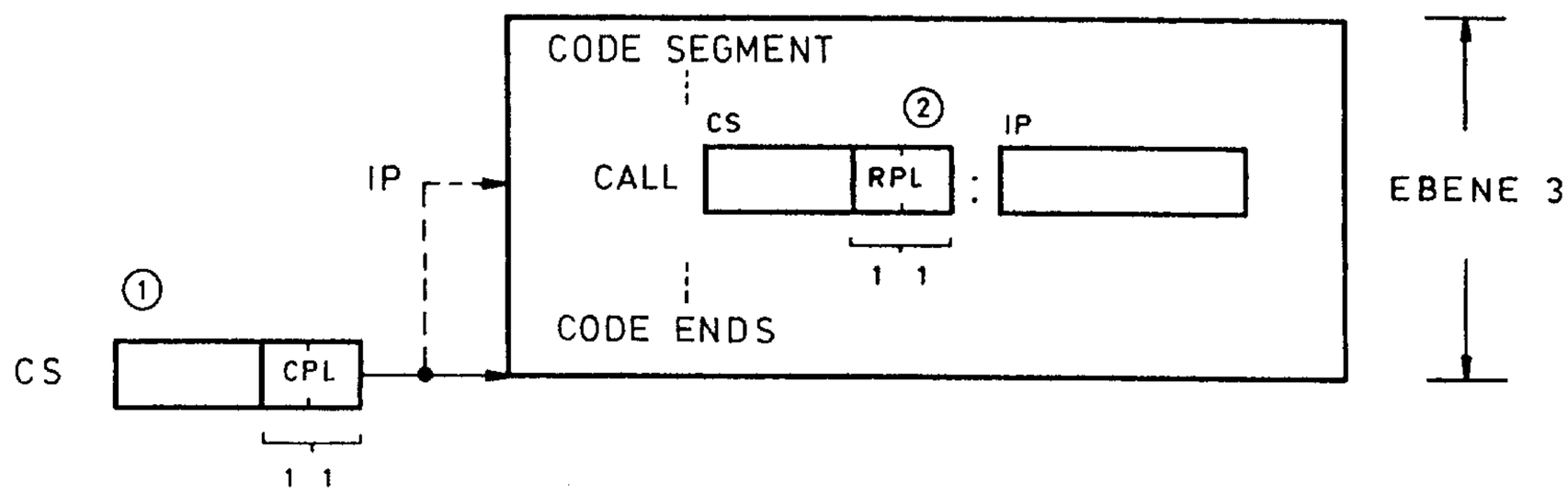
Führt die "Task" das CODE-Segment auf der Privilegebene 3 (CPL = 3) aus, bewirkt der Befehl CALL INTRA keinen Segment-Wechsel, da nur der "Instruction Pointer" IP umgeladen wird.

Im Gegensatz dazu bewirkt der Befehl CALL FAR PTR INTER, daß sowohl das CS-Register als auch IP umgeladen werden. Damit zeigt im CODE-Segment CS:IP nach CODE1:INTER, wodurch ein Segmentwechsel ausgelöst wird. Die "Task" führt nun das CODE1-Segment aus, das sich auf der gleichen Privilegebene 3 (CPL = 3) wie das CODE-Segment befindet.

Aus dem Programm-Listing ist durch die Markierung ---- beim Befehl CALL FAR PTR INTER (CALL Selektor:Offset) ersichtlich, daß ASM286 die Selektorkomponente noch nicht kennt. Diese kann vom CALL-Befehl erst dann geliefert werden, wenn der "System Builder" BLD286 die Systemkonfiguration festgelegt hat.

3.9.3 Requestor's Privilege Level (RPL)

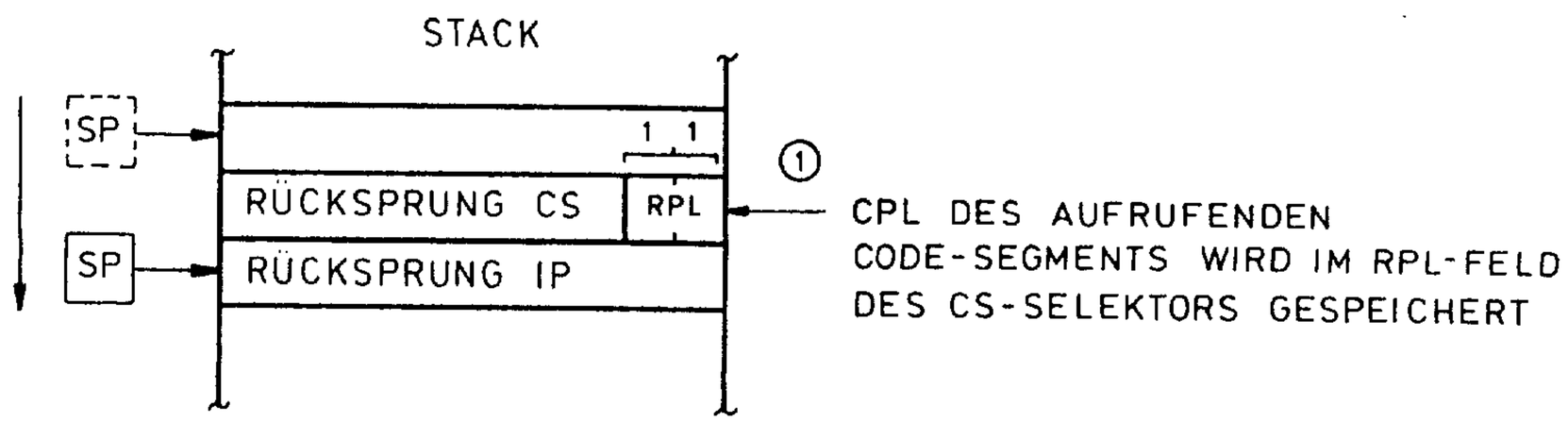
Führt die "Task" das CODE-Segment auf der Privilegebene 3 aus (siehe vorheriges Programmierbeispiel), liefert der Befehl CALL FAR PTR INTER einen Selektor, dessen Privilegebene in seinem RPL-Feld spezifiziert ist. Diese Situation illustriert das folgende Bild:



Vor der Ausführung des CALL-Befehls enthält das RPL-Feld des CS-Selektors die augenblickliche Privilegebene der "Task" (CPL = 3) ①.
Führt die "Task" den CALL-Befehl aus, liefert dieser einen Selektor, dessen Privileg-Attribut RPL "normalerweise" die Ebene des Code-Segments kennzeichnet, die den Selektor generiert hat. Im Beispiel ist dies die Ebene 3 (RPL = 3) ②.

Um eine Verletzung der Schutzmechanismen auszuschließen, darf der Vertrauensgrad des erzeugten Selektors ***nicht größer*** sein als der des aufrufenden Code-Segments. Dieser Vertrauensgrad wird im RPL-Feld des Selektors angezeigt. Im Beispiel gibt das Privileg-Attribut RPL = 3 des vom CALL-Befehl generierten Selektors an, daß sein Vertrauensgrad genauso groß ist wie der des aufrufenden CODE-Segments mit CPL = 3.

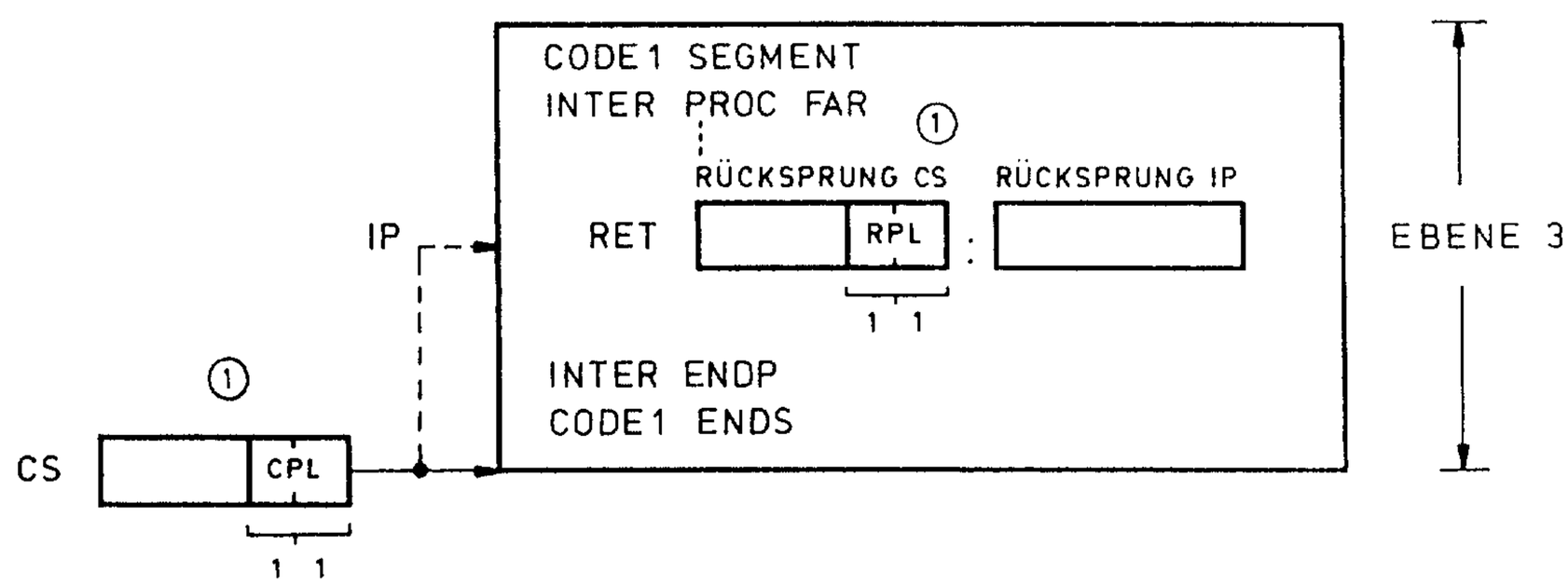
Hat die "Task" den Befehl CALL FAR PTR INTER ausgeführt, steht die Rücksprungadresse zum aufrufenden CODE-Segment im Stack zur Verfügung. Das folgende Bild illustriert diese Situation:



Es ist ersichtlich, daß als Rücksprungadresse sowohl der CS-Selektor als auch der "Instruction Pointer" IP des aufrufenden CODE-Segments im Stack gespeichert ist. Dabei zeigt IP zum ersten Byte nach CALL FAR PTR INTER.

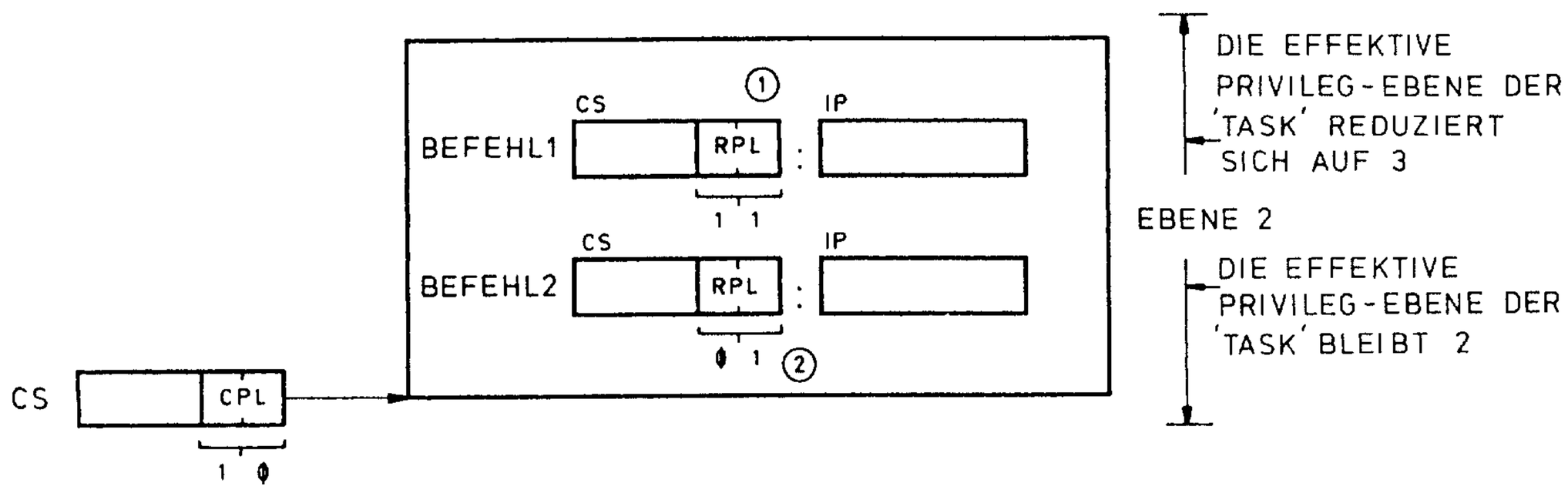
Die Privilegebene der "Task", die das CODE-Segment ausgeführt hat, wird im RPL-Feld des CS-Rücksprung-Selektors gespeichert. Im Beispiel war dies die Ebene 3 ①.

Nachdem der Segment-Wechsel erfolgt ist, führt die "Task" die FAR-Prozedur INTER im CODE1-Segment auf der Privilegebene 3 aus (siehe vorheriges Programmierbeispiel).
Dabei liefert der Befehl RET den CS-Selektor, der vor dem Segment-Wechsel im Stack gespeichert worden ist. Dies soll das folgende Bild verdeutlichen:



Es ist ersichtlich, daß der CS-Selektor mit RPL = 3 den gleichen Vertrauensgrad hat, wie das von der "Task" im Moment ausgeführte CODE1-Segment mit CPL = 3 ①.
Die angegebene Situation liegt nur dann vor, wenn der Befehl RET einen Segment-Wechsel auf der gleichen Privilegebene auslöst.

Der durch RPL spezifizierte Vertrauensgrad eines Selektors kann die effektive Privilegebene einer "Task" niemals verbessern, sondern nur verschlechtern. Dies soll anhand des folgenden Beispiels erläutert werden:



Es sei vorausgesetzt, daß die "Task" ein Code-Segment auf der Privilegebene 2 ausführt (CPL = 2). Liefert z. B. BEFEHL1 einen Selektor mit RPL = 3 ①, reduziert dieser RPL-Wert die effektive Privilegebene der "Task" auf Ebene 3.

Liefert z. B. BEFEHL2 einen Selektor mit RPL = 1 ②, beeinflußt dieser RPL-Wert die effektive Privilegebene der "Task" nicht. Die "Task" bleibt auf der Ebene 2.

Die effektive Privilegebene einer "Task" (EPL = Effective Privilege Level) entspricht daher dem numerischen Maximum von RPL und der augenblicklichen Privilegebene CPL:

EPL = MAX ("Task" CPL, Selektor RPL).

3.10 Ablaufsteuerung eines Programms mit Privileg-Wechsel

Innerhalb einer "Task" kann der Ablauf eines Programms auf ***drei*** Arten gesteuert werden:

- Durch NEAR JMP-, NEAR CALL- und NEAR RET-Befehle innerhalb eines Segments ohne Privileg-Wechsel.
- Durch FAR JMP-, FAR CALL- und FAR RET-Befehle zwischen Segmenten auf der gleichen Privileg-Ebene und
- durch FAR CALL- und FAR RET-Befehle zwischen Segmenten auf unterschiedlichen Privilegebenen.

  (Hinweis: JMP-Befehle auf andere Privilegebenen sind ***nicht*** erlaubt).

Die ersten beiden Typen der Ablaufsteuerung sind bereits diskutiert worden. Es war ersichtlich, daß sie ***keine*** besonderen Steuereinrichtungen benötigten.

Beim dritten Typ der Ablaufsteuerung ist ein spezieller Deskriptor erforderlich, der mittelbar den Wechsel auf eine andere Privilegebene ermöglicht. Dieser spezielle Deskriptor wird als "Call Gate"-Deskriptor bezeichnet und wird von einem FAR CALL-Befehl in der gleichen Weise benützt wie ein Code-Segment-Deskriptor. Allerdings werden die Selektor- und Offsetkomponenten im CALL-Befehl anders behandelt.

Um den Ablauf eines Programms zwischen Segmenten auf verschiedenen Privilegebenen zu steuern, können ***nur*** die folgenden beiden Befehle benützt werden:

- CALL Selektor:Offset
- RET ; (Selektor und Offset werden dem Stack entnom-
  ; men)

Während bei einer Intra-Level-Steuerung mit Segment-Wechsel der Selektor des FAR CALL-Befehls den Deskriptor eines ausführbaren Segments auswählt, selektiert jetzt

- der Selektor einen "Call Gate"-Deskriptor.

Während bei einem "normalen" FAR CALL-Befehl die Offset-Komponente die Einsprungstelle in das Zielsegment markiert, muß auch jetzt

- der Offset-Operand präsent sein, er wird aber ignoriert.

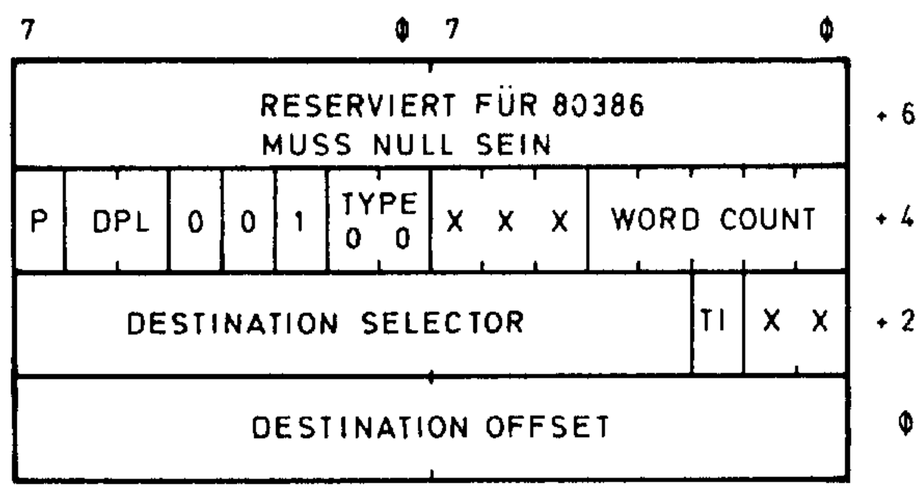
3.10.1 Der "Call Gate"-Deskriptor

Ein "Call Gate"-Deskriptor darf sich entweder

- in der globalen Deskriptor-Tabelle (GDT) oder
- in der lokalen Deskriptor-Tabelle (LDT)

aufhalten, aber ***niemals*** in der Interrupt-Deskriptor-Tabelle (IDT).

Das folgende Bild zeigt das komplette "Layout" eines "Call Gate"-Deskriptors.



Ein Programm ruft mit

- CALL Selektor:Offset
  - Offset → wird ignoriert
  - Selektor → "CALL Gate"-Deskriptor selektieren

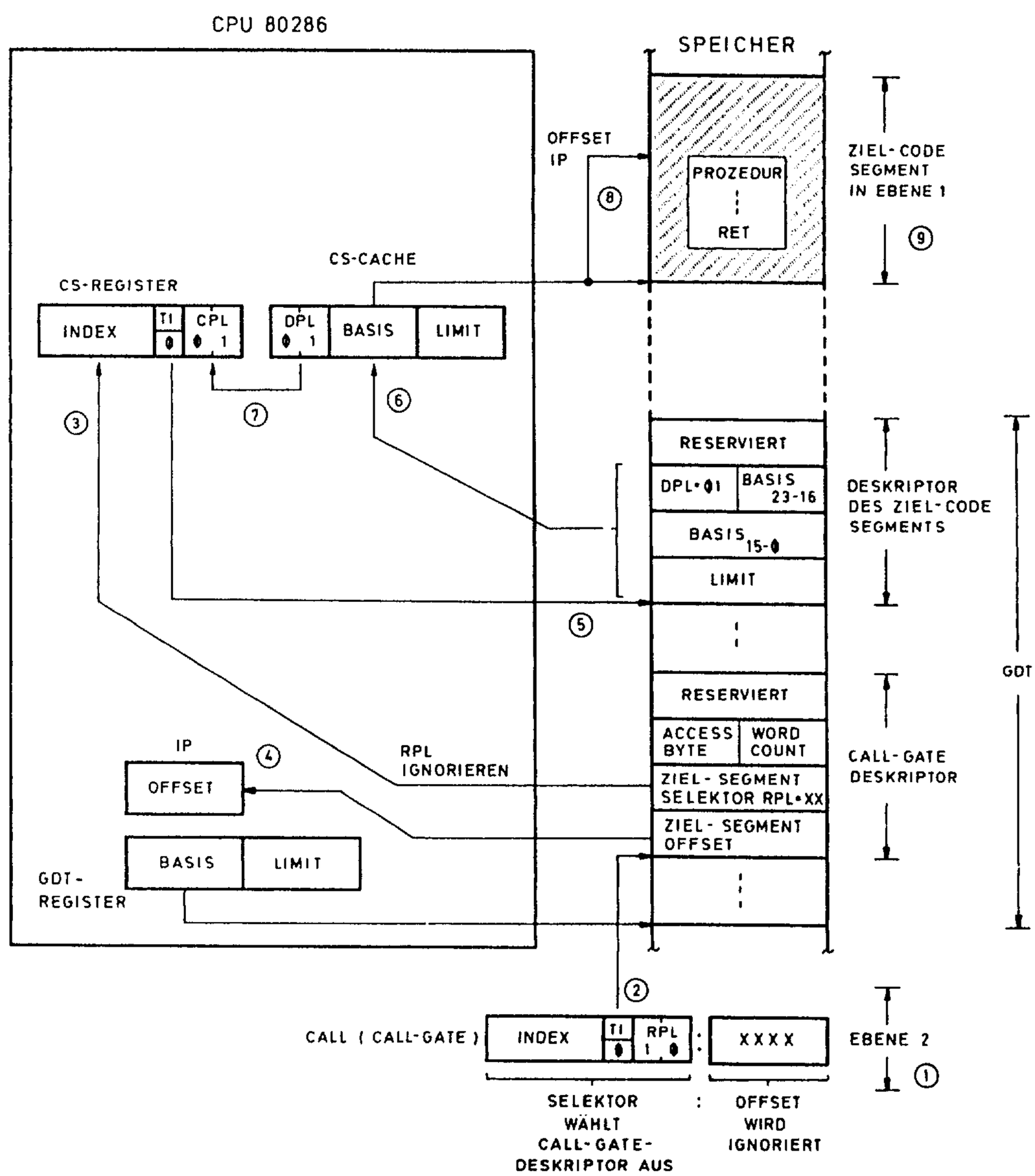
einen "Call Gate"-Deskriptor deswegen auf, um eine ***feste*** Einsprungadresse in

ein Code-Segment in einer höheren Privlegebene zu bekommen. Diese Einsprungsadresse in das Ziel-Segment ist in den Feldern "Destination Selector" und "Destination Offset" im "Call Gate"-Deskriptor enthalten.

Destination-Selector-Feld
Dieses Feld enthält einen Selektor, der entweder in der GDT (TI = 0) oder in der LDT (TI = 1) einen Deskriptor für ein ausführbares Code-Segment auswählt. Das RPL-Feld in diesem Selektor wird ***nicht*** benützt (xx).

Destination-Offset-Feld
Dieses Feld enthält einen Zeiger, der die Anfangsadresse der auszuführenden Prozedur im ausgewählten Ziel-Code-Segment kennzeichnet.

Das folgende Bild zeigt die Analyse eines Privileg-Wechsels über einen "Call Gate"-Deskriptor.



Es sei vorausgesetzt, daß die "Task" ein Code-Segment auf der Privilegebene 2 ausführt ①. In diesem Segment soll ein CALL-Befehl einen Selektor liefern (RPL = 2), der in der GDT (TI = 0) einen "Call Gate"-Deskriptor auswählt ②.

Der im "Call Gate"-Deskriptor gespeicherte Ziel-Segment-Selektor überschreibt das augenblickliche CS-Register, wobei RPL dieses Selektors ignoriert wird ③. Parallel dazu wird der Ziel-Segment-Offset in den "Instruction Pointer" IP geladen ④.

Der im CS-Register gespeicherte Selektor wählt nun in der GDT (TI = 0) den Deskriptor des Ziel-Code-Segments aus ⑤, der anschließend in das CS-Cache-Register geladen wird ⑥.

Die Privilegebene des Ziel-Code-Segments sei 1. Diese Ebene ist im DPL-Feld des zugehörigen Deskriptors spezifiziert. Nach der Initialisierung des CS-Cache-Registers mit dem Deskriptor des Ziel-Code-Segments wird der DPL-Wert des Deskriptors in das RPL-Feld des CS-Selektors kopiert ⑦. Dadurch wird im CS-Selektor angezeigt, daß sich die "Task" jetzt auf der Privilegebene 1 befindet (CPL = 1), was dem gewünschten Privileg-Wechsel von Ebene 2 nach Ebene 1 entspricht.
Die Einsprungstelle der Prozedur im Ziel-Code-Segment entnimmt die "Task" dem "Instruction Pointer" IP ⑧. Die "Task" führt nun das Ziel-Code-Segment auf der Privilegebene 1 aus ⑨.

3.10.2 Programmierbeispiel:
Privileg-Wechsel über einen "Call Gate"-Deskriptor

Um einen Privileg-Wechsel zu realisieren, ist der Aufruf eines "Call Gate"-Deskriptors erforderlich. Das folgende Programm-Fragment enthält die hierfür notwendige Codierung.

```
iAPX286 MACRO ASSEMBLER    B10

PC 16-20 - iAPX286 MACRO ASSEMBLER VX.X ASSEMBLY OF MODULE B10
OBJECT MODULE PLACED IN B10.OBJ
ASSEMBLER INVOKED BY:  ASM286.86 B10.SRC


LOC  OBJ                  LINE     SOURCE

                            1              NAME    B10
                            2
                            3     EXTRN CALL_GATE:FAR
                            4
....                        5     EBENE_2 SEGMENT EO                    ;Nur ausfuehrbares Segment mit
                            6                                           ;Basis-Adresse=1000H und DPL=2
```

```
                          7                                        ;(Festlegung durch BLD286)
                          8            ;
                          9            ;                           ;Normale Befehlssequenz
                         10            ;
0000 9A0000----     E    11            CALL CALL_GATE              ;Aufruf der Prozedur INTER_LEVEL
                         12                                        ;in der Privileg-Ebene 1 ueber einen
                         13                                        ;Call Gate-Deskriptor
                         14                                        ;(Festlegung durch BLD286)
                         15            ;
                         16            ;                           ;Normale Befehlssequenz
----                     17     EBENE_2 ENDS
                         18
                         19
----                     20     EBENE_1 SEGMENT EO                  ;Nur ausfuehrbares Segment mit
                         21                                        ;Basisadresse=2000H und DPL=1
                         22                                        ;(Festlegung durch BLD286)
                         23            ;
                         24            ;                           ;Normale Befehlssequenz
                         25            ;
0000                     26     INTER_LEVEL  PROC   FAR  ;*********************************************************
                         27            ;
                         28            ;                           ;Prozedurkoerper
                         29            ;
0000 CB                  30            RET
                         31     INTER_LEVEL  ENDP        ;*********************************************************
                         32            ;
                         33            ;                           ;Normale Befehlssequenz
----                     34     EBENE_1 ENDS
                         35
                         36            END

ASSEMBLY COMPLETE,   NO WARNINGS,   NO ERRORS
```

Es ist ersichtlich, daß der Modul die beiden Code-Segmente EBENE_1 auf der Privilegebene 1 (DPL = 1) und EBENE_2 auf der Privilegebene 2 (DPL = 2) enthält. Beide Segmente werden von Deskriptoren beschrieben, die der "System Builder" BLD286 spezifiziert hat.

Wenn die "Task" das EBENE_2-Segment auf der Privilegebene 2 ausführt, liefert der Befehl CALL CALL_GATE einen Selektor, der abhängig vom Wert seines "Table Indicators" TI einen "Call Gate"-Deskriptor in der GDT (TI = 0) oder in der LDT (TI = 1) auswählt.

Im ASM286-Listing ist dieser Selektor durch ---- markiert.

Für die Offset-Komponente liefert CALL CALL_GATE den Wert 0000, da der Offset-Operand zwar präsent sein muß, aber nicht ausgewertet wird.

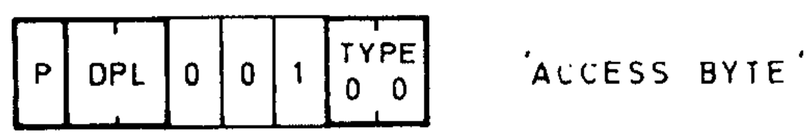
CALL_GATE ist der Name eines "Call Gate"-Deskriptors. Dieser Deskriptor wird vom "System Builder" BLD286 spezifiziert und durch frei wählbare Namen (im Beispiel CALL_GATE) identifiziert. Die von BLD286 generierte Systemkomposition ist in einem eigenen Modul, in der sogenannten "Build File", enthalten. Dies bedeutet, daß in dieser "File" auch der Deskriptor CALL_GATE existiert.

Daher ist dem Assembler durch EXTRN CALL_GATE:FAR mitzuteilen, daß der Deskriptor CALL_GATE in einer externen Datei definiert ist.

Der im Deskriptor CALL_GATE enthaltene Selektor wählt den Deskriptor des EBENE_1-Segments aus, während die in CALL_GATE enthaltene Offsetkomponente die FAR-Prozedur INTER_LEVEL markiert.

3.10.3 "Call Gate"-Deskriptor Privileg-Level (DPL)

Das folgende Bild zeigt das "Access Byte" des "Call Gate"-Deskriptors:



Es ist ersichtlich, daß das "Access Byte" ebenfalls ein DPL-Feld enthält, dessen Wert die Privilegebene angibt, auf der sich der "Call Gate"-Deskriptor befindet.

Das "Type"-Feld
Der 80286 kennt ***vier*** Arten von "Gate"-Deskriptoren, deren Typ im "Type"-Feld des Deskriptors spezifiziert wird. Diese "Gate"-Deskriptoren sind

- das "Call Gate" ; Type = 00
- das "Task Gate" ; Type = 01
- das "Interrupt Gate" ; Type = 10
- das "Trap Gate" ; Type = 11

In diesem Kapitel soll zunächst nur das "Call Gate" betrachtet werden.

Das "Present"-Bit (P)
Da sich ein "Gate"-Deskriptor ***nicht*** direkt auf ein "normales" Segment bezieht, ist es nicht notwendig, daß das "Present"-Bit im "Gate"-Deskriptor anzeigt, ob ein Segment gegenwärtig ist. Das P-Bit kann jedoch für andere Zwecke benützt werden.

Ist ein Basis-System mit BLD286 konfiguriert, müssen Systemdienste wie Ein-/Ausgaben oder Routinen zur Behandlung von Ausnahmesituationen für Anwenderprogramme verfügbar sein.

Daher ist es notwendig, daß System-Codes über "Gate"-Deskriptoren erreichbar sind. Diese vom Anwender spezifizierten "Gate"-Deskriptoren können vom "System Builder" BLD286 in sogenannte "Export Files" ausgelagert und bei Bedarf mit dem Binder BND286 in das System eingebunden werden.

Die "Export Files" kann nun der Anwender benützen, um die Schnittstellen zwischen Anwender-Software und System-Software herzustellen.

Um dem Basis-System mitzuteilen, daß die "Gate"-Deskriptoren ausgelagert sind, muß BLD286 diese Deskriptoren als ***NOT PRESENT*** (P-Bit = 0) markieren. Damit ist der Inhalt der "Call Gate"-Deskriptoren als ungültig gekennzeichnet.

3.10.4 Regeln für "Interlevel"-Programmabläufe

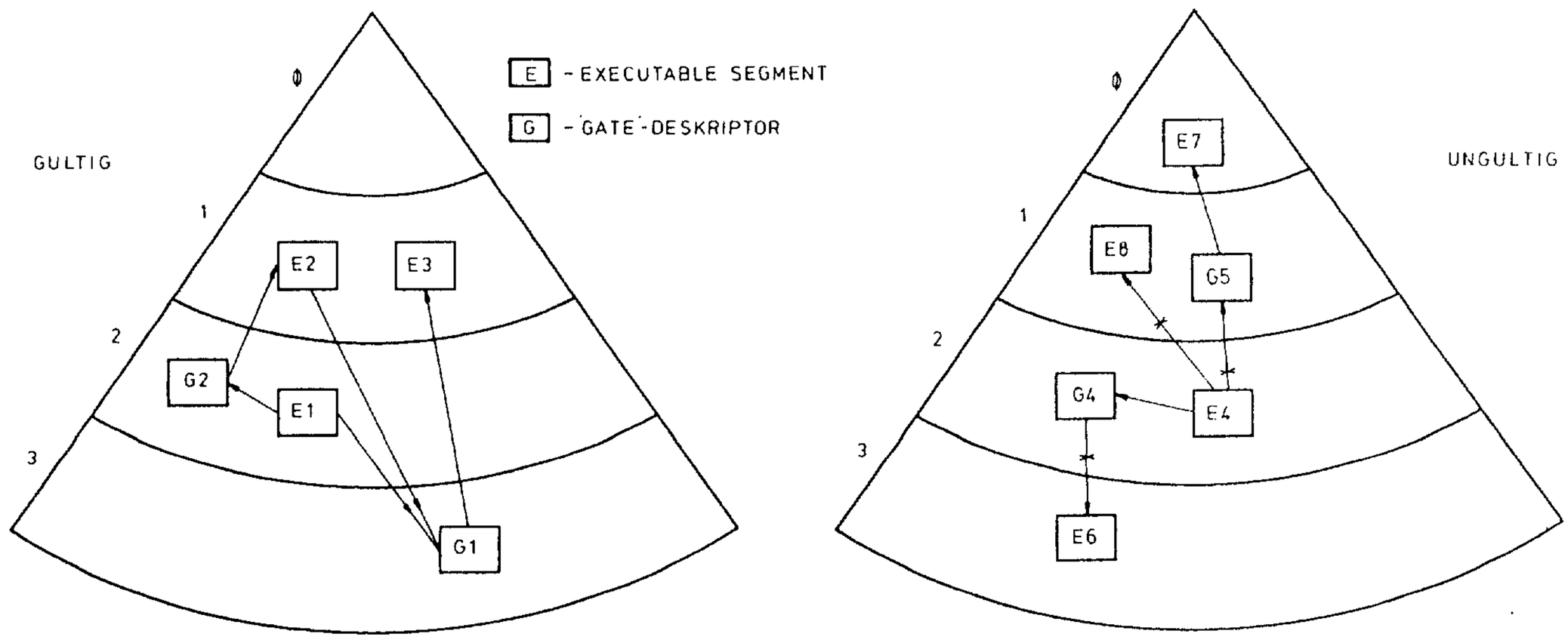
Bei einem Privileg-Wechsel über einen "Gate"-Deskriptor sind vier Privilegebenen-Felder von Bedeutung.
Diese sind:

- Die augenblickliche Privilegebene (CPL) des augenblicklich ausgeführten Segments.
- Die Privilegebene (RPL) des vom CALL-Befehl benützten Selektors.
- Die Privilegebene (DPL) des "Gate"-Deskriptors und
- die Privilegebene (DPL) im Segment-Deskriptor des auszuführenden Ziel-Code-Segments.

Ein Segment-Wechsel zwischen verschiedenen Privilegebenen ("Interlevel"-Programmablauf) ist nur dann realisierbar, wenn die folgenden Beziehungen zwischen den Kennziffern der Privilegebenen eingehalten werden:

MAX ("Task" CPL, Selektor RPL)	≤ **"Gate" DPL**
und	
Zielsegment DPL	≤ **"Task" CPL**

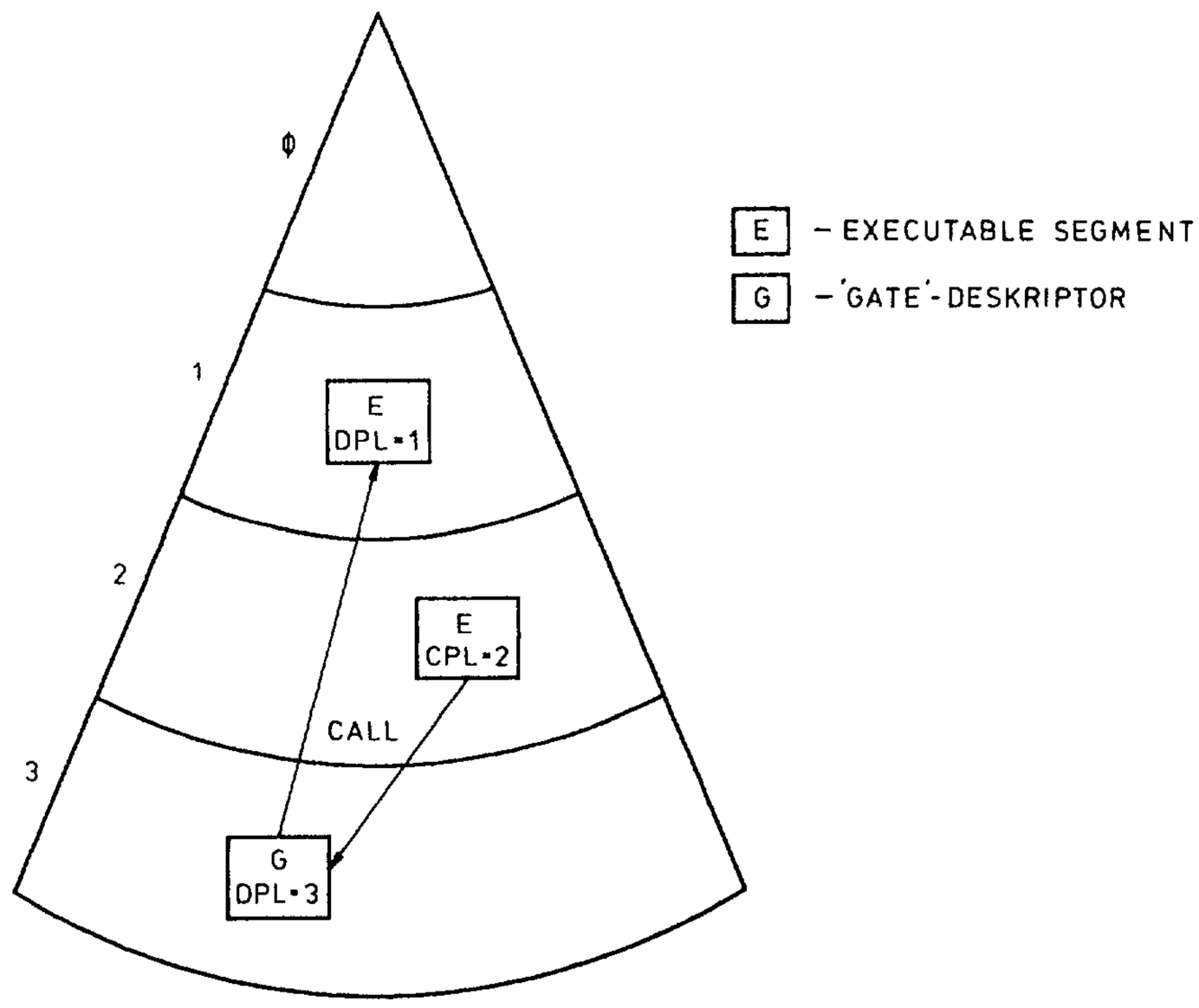
Die folgenden Bilder illustrieren sowohl gültige als auch ungültige Versuche, einen "Interlevel"-Programmablauf zu realisieren.



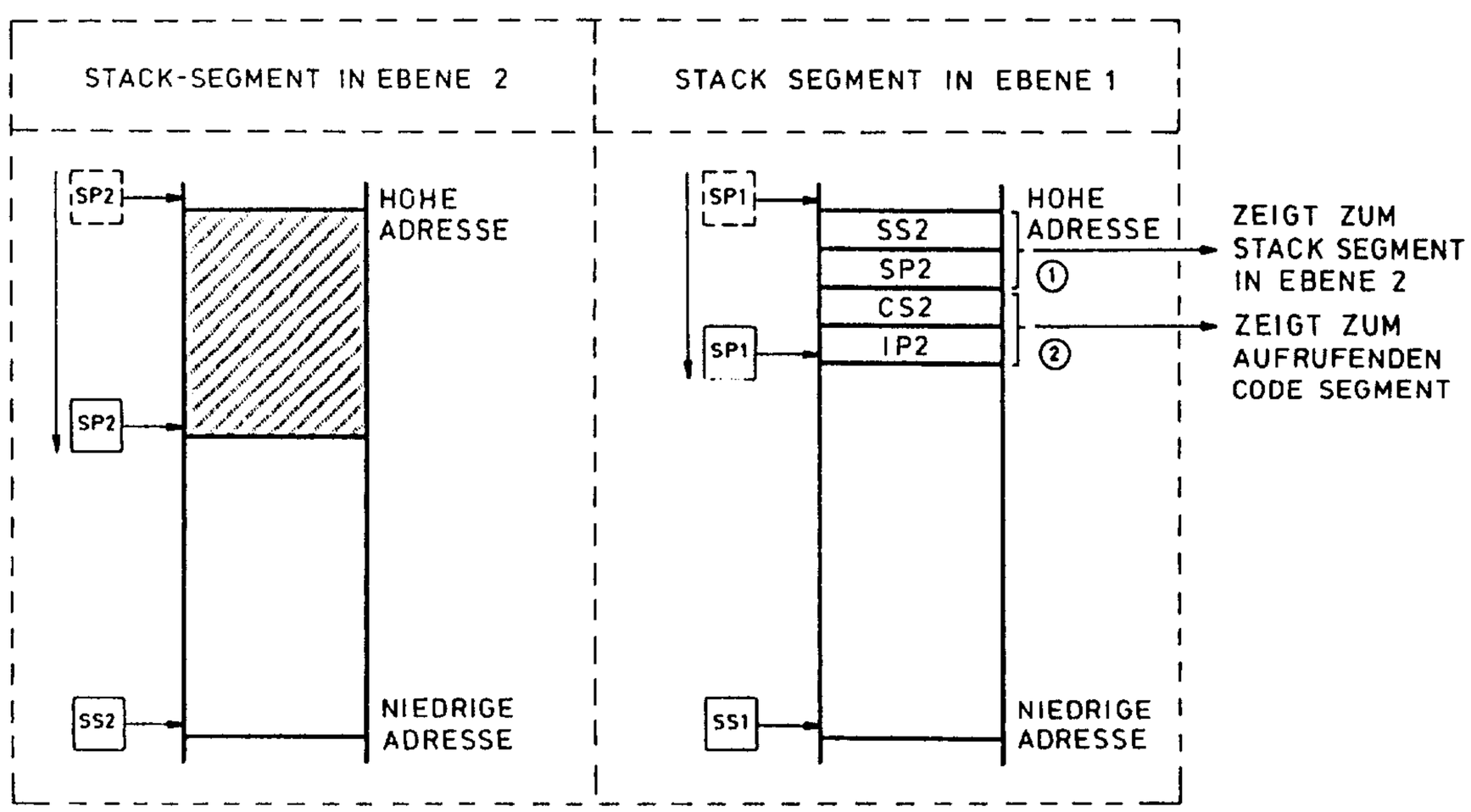
- Der Pfad ***E4-G5-E7*** ist ***ungültig,*** da die Privilegebene DPL des "Gate"-Deskriptors G5 numerisch kleiner ist als CPL des Code-Segments E4.
- Der Pfad ***E4-E8*** ist ebenfalls ***ungültig,*** da ein Privileg-Wechsel nur über einen "Gate"-Deskriptor möglich ist.
- Auch der Pfad ***E4-G4-E6*** ist ***ungültig,*** da die Privilegebene des Zielsegments E6 numerisch größer ist als CPL des aktiven Code-Segments E4.
- Nur die Pfade ***E1-G2-E2, E1-G1-E3*** und ***E2-G1-E3*** sind ***gültig,*** da hier die oben angegebenen Regeln befolgt werden.

3.11 "Interlevel"-Rückkehr

Immer dann, wenn über einen "Call Gate"-Deskriptor ein Code-Segment in einer höheren Privilegebene aufgerufen wird, bewirkt der Privileg-Wechsel, daß automatisch auch das Stack-Segment gewechselt wird.
Diese Situation sollen die folgenden Bilder an einem Beispiel verdeutlichen.



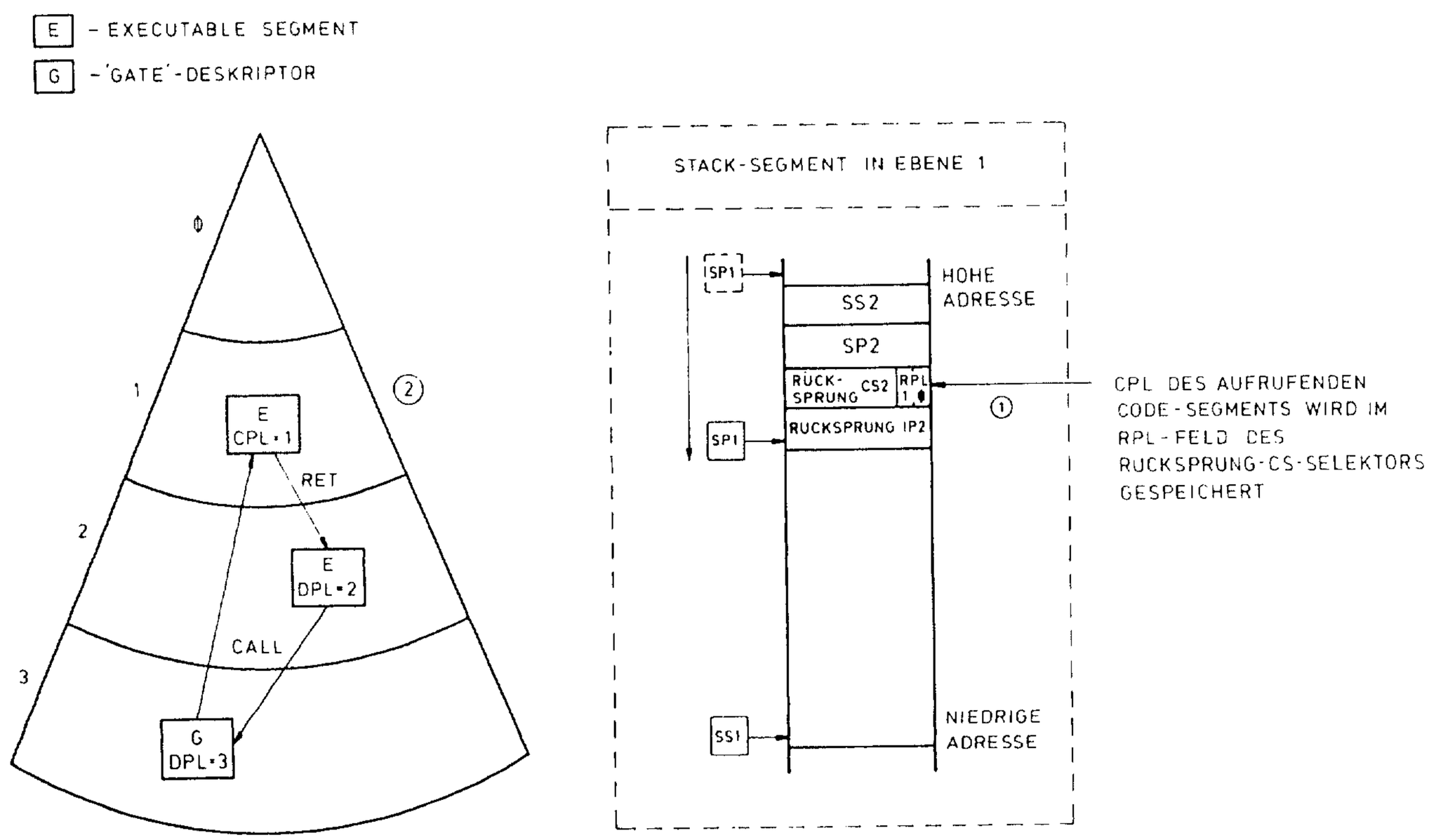
- Es sei vorausgesetzt, daß ein Code-Segment auf der Privilegebene 2 (CPL = 2) über einen "Call Gate"-Deskriptor auf der Privilegebene 3 (DPL = 3) ein Code-Segment auf der Privilegebene 1 (DPL = 1) aufruft.



Die Ausführung des CALL-Befehls auf der Privilegebene 2 (CPL = 2) bewirkt, daß zunächst der momentane 32-Bit-Zeiger SS2:SP2 zum Stack-Segment in Ebene 2 an das Stack-Segment in Ebene 1 übergeben wird ①. Dabei ist das Ebene-1-Stack-Segment durch einen eigenen 32-Bit-Zeiger SS1:SP1 gekennzeichnet.

Die Rücksprungadresse CS2:IP2 zum aufrufenden Code-Segment in Ebene 2 wird anschließend im Ebene-1-Stack-Segment gespeichert ②. Dabei zeigt IP2 zum nächsten Befehl nach CALL.

Wie bereits früher erwähnt, wird im RPL-Feld des Rücksprung-CS-Selektors die augenblickliche Privilegebene (CPL) des aufrufenden Code-Segments gespeichert. Im Beispiel ist dies die Ebene 2 ①.



Wenn die "Task" das Ziel-Code-Segment ausführt, befindet sie sich auf der Privilegebene 1 (CPL = 1). Der RET-Befehl in diesem Segment ② liefert den im Ebene-1-Stack gespeicherten Rücksprung-Selektor CS2 und den Rücksprung-"Instruction Pointer" IP2. Dabei hat der Rücksprung-Selektor CS2 die Privilegebene 2 (RPL = 2).

Immer dann, wenn der RET-Befehl auf einen CS-Wert trifft, dessen RPL > CPL ist, wird ein sogenannter "Interlevel"-Rücksprung eingeleitet. Dies ist im angegebenen Beispiel mit RPL = 2 und CPL = 1 der Fall.

Da RPL des Rücksprung-Selektors CS2 numerisch größer ist als CPL, wird die effektive Privilegebene der "Task" EPL = MAX ("Task" CPL, Selektor RPL) verschlechtert.

Das angegebene Beispiel illustriert, daß ein Privileg-Wechsel durch einen RET-Befehl nur dann möglich ist, wenn die augenblickliche Privilegebene CPL, in der sich der RET-Befehl befindet, numerisch ***kleiner*** ist als DPL des Code-Segments, zu dem der Rücksprung erfolgen soll. Dabei identifiziert RPL im Rücksprung-CS-Selektor die angeforderte Rücksprung-Privilegebene.

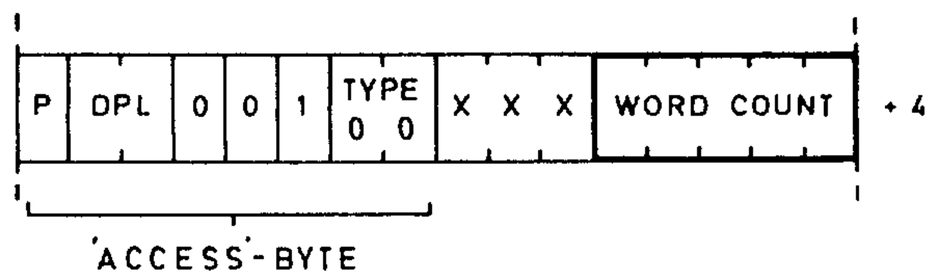
3.12 "Interlevel"-Prozeduraufruf mit Parameterübergabe

3.12.1 Das "WORD COUNT"-Feld

Wenn ein "Call Gate"-Deskriptor benützt wird, um Privilegebenen zu wechseln, muß das jeweils neue Stack-Segment genügend Platz für die Rücksprungadresse, für alle übergebenen Parameter und die lokalen Variablen enthalten.

Um den Privileg-Wechsel für das aufrufende Programm "durchsichtiger" zu machen, gibt ein "Call Gate"-Deskriptor an, daß Parameter vom ***alten*** Stack in den ***neuen*** Stack kopiert worden sind.

Wieviele Worte (bis zu 31) vom Stack des aufrufenden Programms in den neuen Stack kopiert wurden, spezifiziert das "WORD COUNT"-Feld im "Call Gate"-Deskriptor. Enthält das "WORD COUNT"-Feld den Wert Null, wurden ***keine*** Parameter kopiert.
Das folgende Bild zeigt das "WORD COUNT"-Feld im "Call Gate"-Deskriptor:



3.12.2 **Programmierbeispiel:** Parameterübergabe

Anhand des folgenden Programm-Fragments soll gezeigt werden, wie in ASM286 der "WORD COUNT"-Parameter zu benützen ist und welche Stack-Layouts sich vor und nach dem Privileg-Wechsel ergeben.

```
iAPX286 MACRO ASSEMBLER    B11

PC 16-20 - iAPX286 MACRO ASSEMBLER VX.X ASSEMBLY OF MODULE B11
OBJECT MODULE PLACED IN B11.OBJ
ASSEMBLER INVOKED BY:  ASM286.86 B11.SRC


LOC  OBJ                  LINE     SOURCE

                             1             NAME    B11
                             2
                             3     ASSUME DS:EBENE_2_DATEN
                             4
                             5     EXTRN CALL_GATE:FAR
                             6
----                         7     EBENE_2_DATEN  SEGMENT RW              ;Les- und schreibbares Segment mit
                             8                                            ;Basis-Adresse=1000H und DPL=2
                             9                                            ;(Festlegung durch BLD286)
0000 ????                   10     PARAMETER1  DW      ?
0002 ????                   11     PARAMETER2  DW      ?
0004 ????                   12     PARAMETER3  DW      ?
                            13
----                        14     EBENE_2_DATEN  ENDS
                            15
----                        16     EBENE_2_CODE  SEGMENT EO               ;Nur ausfuehrbares Segment mit
                            17                                            ;Basis-Adresse=2000H und DPL=2
                            18                                            ;(Festlegung durch BLD286)
                            19             ;
                            20             ;                              ;Normale Befehlssequenz
                            21             ;
0000 FF360000               22             PUSH    PARAMETER1             ;Uebergabe von 3 Word-Parametern
0004 FF360200               23             PUSH    PARAMETER2             ;an das Stack-Segment in Ebene 1
0008 FF360400               24             PUSH    PARAMETER3
000C 9A0000----      E      25             CALL CALL_GATE                 ;Aufruf der Prozedur INTER_LEVEL
                            26                                            ;in der Privileg-Ebene 1 ueber einen
                            27                                            ;Call Gate-Deskriptor
                            28                                            ;(Festlegung durch BLD286)
                            29             ;
                            30             ;                              ;Normale Befehlssequenz
----                        31     EBENE_2_CODE  ENDS
                            32
                            33
----                        34     EBENE_1_CODE  SEGMENT EO               ;Nur ausfuehrbares Segment mit
                            35                                            ;Basisadresse=3000H und DPL=1
                            36                                            ;(Festlegung durch BLD286)
                            37             ;
                            38             ;                              ;Normale Befehlssequenz
                            39             ;
0000                        40     INTER_LEVEL  PROC   FAR  WC(3) ;*********************************************
                            41             ;
```

```
                        42          ;                          ;Prozedurkoerper
                        43          ;
0000 CA0600             44          RET 6                      ;Entferne die Parameter
                        45    INTER_LEVEL  ENDP            ;*********************************************
                        46          ;
                        47          ;                          ;Normale Befehlssequenz
....                    48    EBENE_1_CODE  ENDS
                        49
                        50          END
ASSEMBLY COMPLETE,   NO WARNINGS,   NO ERRORS
```

Das Modul enthält das les- und schreibbare Datensegment EBENE_2_DATEN (Typ RW) auf der Privilegebene 2 und die beiden ausführbaren Code-Segmente EBENE_2_CODE auf der Privilegebene 2 und EBENE_1_CODE auf der Privilegebene 1 (Typ EO).

Im EBENE_2_DATEN-Segment sind drei Parameter (PARAMETER 1 bis 3) definiert, die zunächst im EBENE_2_CODE-Segment mit PUSH-Befehlen in den Stack der ***Privilegebene 2*** übergeben werden.
Anschließend erfolgt mit CALL CALL_GATE der Aufruf der FAR-Prozedur INTER_LEVEL in der Privilegebene 1 über einen "Call Gate"-Deskriptor.

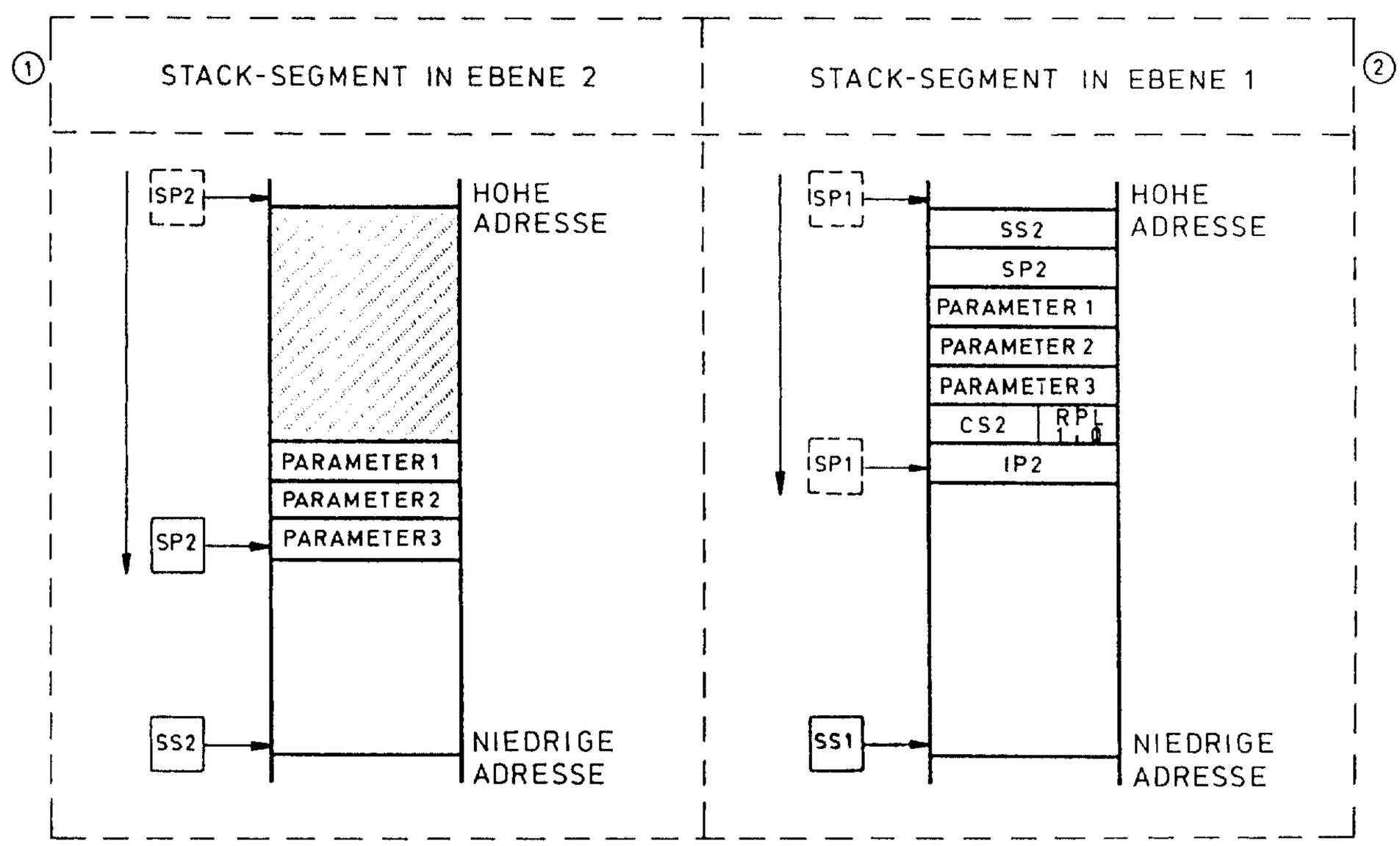
Es ist ersichtlich, daß in der INTER_LEVEL-Prozedurvereinbarung der "***W***ORD ***C***OUNT"-Parameter WC(3) angegeben ist, der die allgemeine Form ***WC (Ausdruck)*** hat. Dabei ist ***Ausdruck*** ein Integerwert im Bereich von 0 bis 31. Dieser Wert spezifiziert die Anzahl der Worte, die beim Aufruf der privilegierteren Prozedur als Parameter in den privilegierteren Stack kopiert werden sollen.

Wie das Programm-Fragment zeigt, wird der "WORD COUNT"-Parameter nicht nur im "Call Gate"-Deskriptor benützt, sondern auch in der PROC-Direktive:
Dies hat den Vorteil, daß damit der Anwender über die Anzahl der kopierten Parameter informiert werden kann, da für ihn der "Call Gate"-Deskriptor "unsichtbar" ist.

Vor dem Aufruf der Prozedur INTER_LEVEL durch CALL CALL_GATE ergibt sich wegen

- PUSH PARAMETER1
- PUSH PARAMETER2
- PUSH PARAMETER3

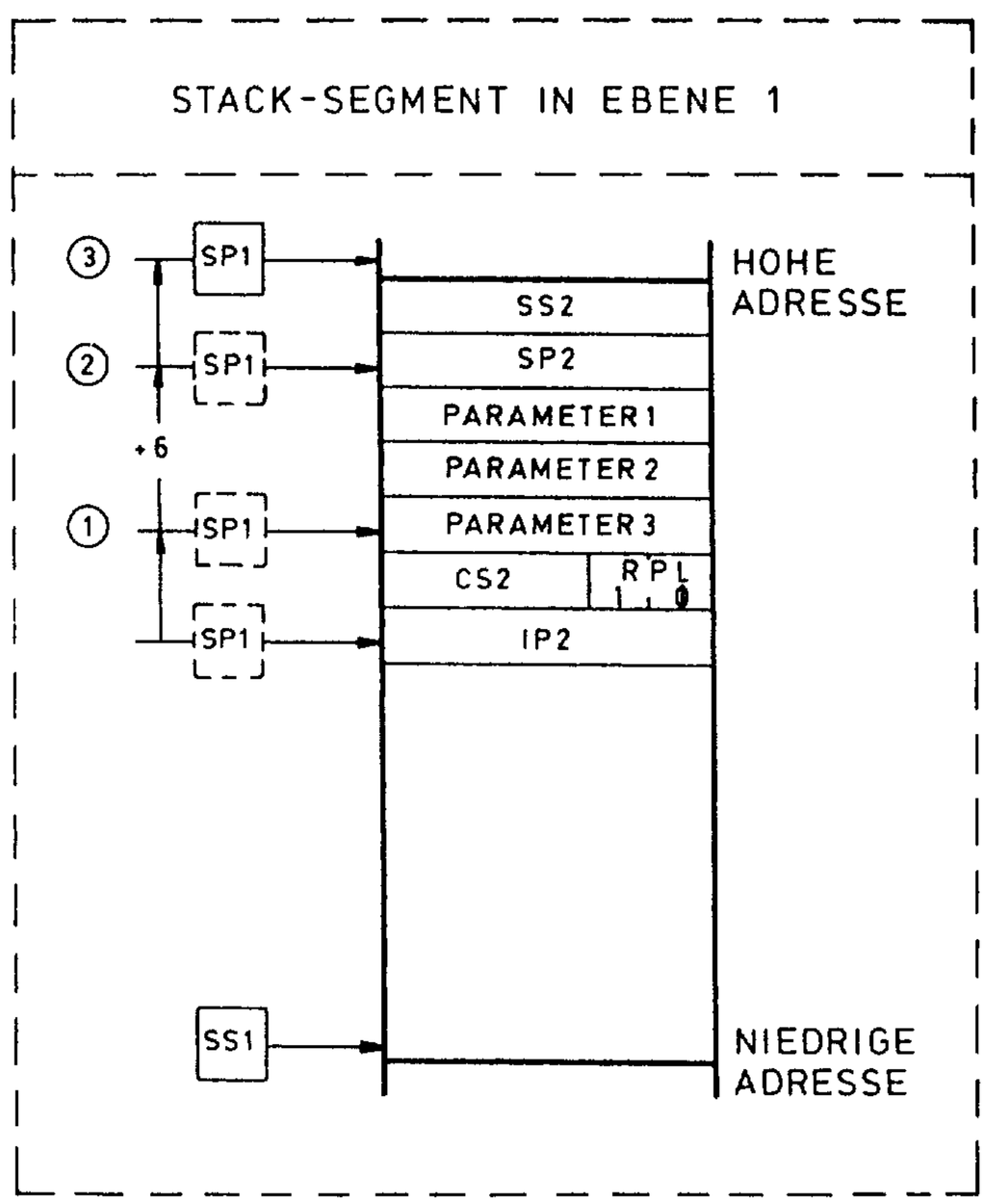
folgendes Stack-Layout in der Privilegebene 2 ①.



Wird nun wegen CALL CALL_GATE der Privileg-Wechsel von Ebene 2 nach Ebene 1 durchgeführt, bestimmt der Wert im "WORD COUNT"-Feld des "Call Gate"-Deskriptors, wieviele Worte vom Ebene-2-Stack in den Ebene-1-Stack kopiert werden sollen.

Ist der Privileg-Wechsel erfolgt, ergibt sich in der Privilegebene 1 das oben angegebene Stack-Layout ②.
Es ist ersichtlich, daß zunächst der Zeiger SS2:SP2 zum vorherigen Stack im aktuellen Stack gespeichert wird. Anschließend werden die Parameter, deren Anzahl im "WORT COUNT"-Feld spezifiziert ist, vom alten in den neuen Stack kopiert.
Schließlich wird danach der Zeiger CS2:IP2 zum aufrufenden Code-Segment im neuen Stack gespeichert.

Am Ende der FAR-Prozedur INTER_LEVEL erfolgt mit RET 6 der Rücksprung in das aufrufende EBENE_2_CODE-Segment. Die Funktion des Befehls RET 6 soll anhand des Stack-Layouts der Privilegebene 1 erläutert werden.



Der Rücksprung zum EBENE_2_CODE-Segment wird realisiert, indem RET 6 den "Instruction Pointer" IP mit dem IP2-Wert und das CS-Register mit dem CS2-Wert initialisiert. Danach zeigt SP1 zum Parameterfeld ①.

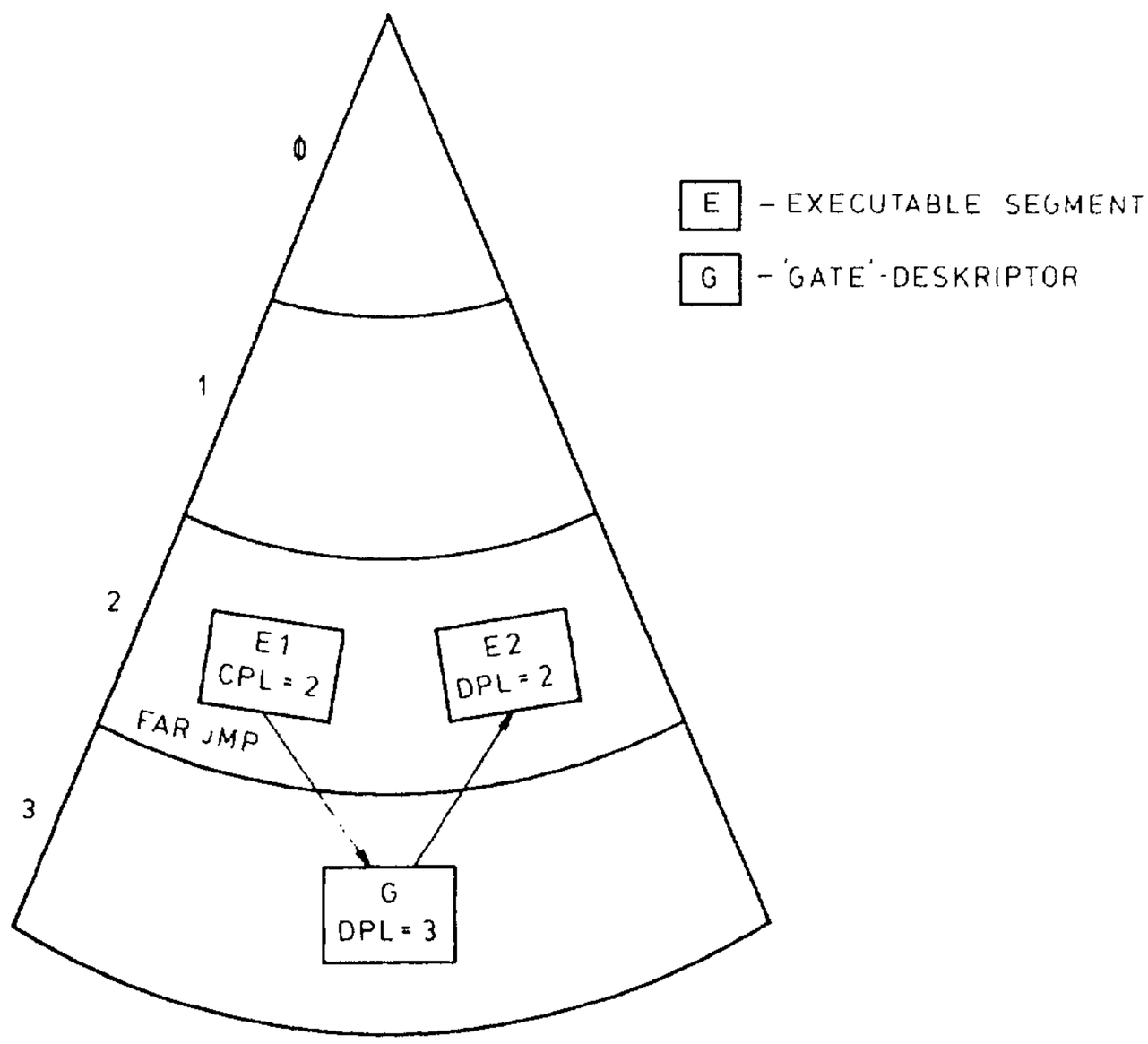
Um nun die Parameter aus dem Ebene-1-Stack zu "entfernen", ist es notwendig, SP1 auf eine Position jenseits des Parameter-Feldes zu stellen. Im Beispiel wird dies erreicht, wenn der Offsetwert in SP1 um 6 (6 Bytes = 3 Wortparameter) erhöht wird. Dies leistet der Operand im Befehl RET 6 automatisch.
Ist RET 6 ausgeführt, zeigt SP1 zum 32-Bit-Zeiger SS2:SP2, der das Stack-Segment in der Ebene 2 kennzeichnet ②.
Dieser Zeiger wird nun in das SS-Register und das SP-Register zurückgeladen, so daß SP1 wieder an seiner ursprünglichen Position steht ③.

Es sei an dieser Stelle darauf hingewiesen, daß die Stack-Zeiger SS0:SP0, SS1:SP1 und SS2:SP2 für die Privilegebenen 0, 1 und 2 ***nicht*** von CPU-Registern, sondern von einem speziellen System-Segment ("Task"-Status-Segment) geliefert werden. Dies wird im Kapitel "Tasks" und Status-Wechsel näher erläutert.

3.13 JMP-Befehl und "Call Gate"-Deskriptor

Wie bereits früher erwähnt, kann ein JMP-Befehl für einen Privileg-Wechsel ***nicht*** benützt werden. Er steht lediglich zur Steuerung eines Intra-Segment- oder eines Inter-Segment-Programmablaufs auf der gleichen Privilegebene zur Verfügung.

Für den Fall, daß durch einen JMP-Befehl ein Segmentwechsel erfolgen soll, kann zur Auflage gemacht werden, daß nicht "wild" in das Ziel-Code-Segment gesprungen wird. Dies wird erreicht, wenn der JMP-Befehl einen "Call Gate"-Deskriptor benützt, da dieser Deskriptor eine feste Einsprungadresse definiert. Das folgende Bild illustriert eine solche Situation.



Es ist ersichtlich, daß im E1-Code-Segment der Befehl

- JMP Selektor:Offset (FAR JMP)
  - Offset → wird ignoriert
  - Selektor → "Call-Gate"-Deskriptor selektieren

ausgeführt wird, dessen Selektorkomponente einen "Call Gate"-Deskriptor auf der Privilegebene 3 auswählt. Dieser Deskriptor definiert die Einsprungadresse in das E2-Code-Segment, das sich mit dem aufrufenden Code-Segment E1 auf der gleichen Privilegebene befinden muß.

Wie das Beispiel zeigt, kann ein FAR JMP-Befehl einen "Call Gate"-Deskriptor nur zur Steuerung eines Intersegment-Programmablaufs auf der gleichen Privilegebene benützen.
Ein Segmentwechsel durch JMP über einen "Call Gate"-Deskriptor ist immer dann realisierbar, wenn folgende Beziehungen zwischen den Kennziffern der Privilegebenen eingehalten werden:

MAX ("Task" CPL, Selektor RPL)	**≤ "Gate" DPL**
und	
Zielsegment DPL	**= "Task" CPL**

3.14 Daten-Zugriffe

Auf Daten darf entweder in ***Daten***-Segmenten oder in ***lesbaren*** Code-Segmenten zugegriffen werden. Immer dann, wenn die Register DS, SS oder ES z. B. durch die Befehle LDS, LES oder MOV geladen werden, erfolgt eine Überprüfung des "Access"-Bytes im zugehörigen Deskriptor.
Dabei wird festgestellt,

- ob der zu ladende Deskriptor-Typ zum Segment-Register paßt und
- ob ein Segment-Register den Zugriff zu einem bestimmten Segment-Typ zuläßt.

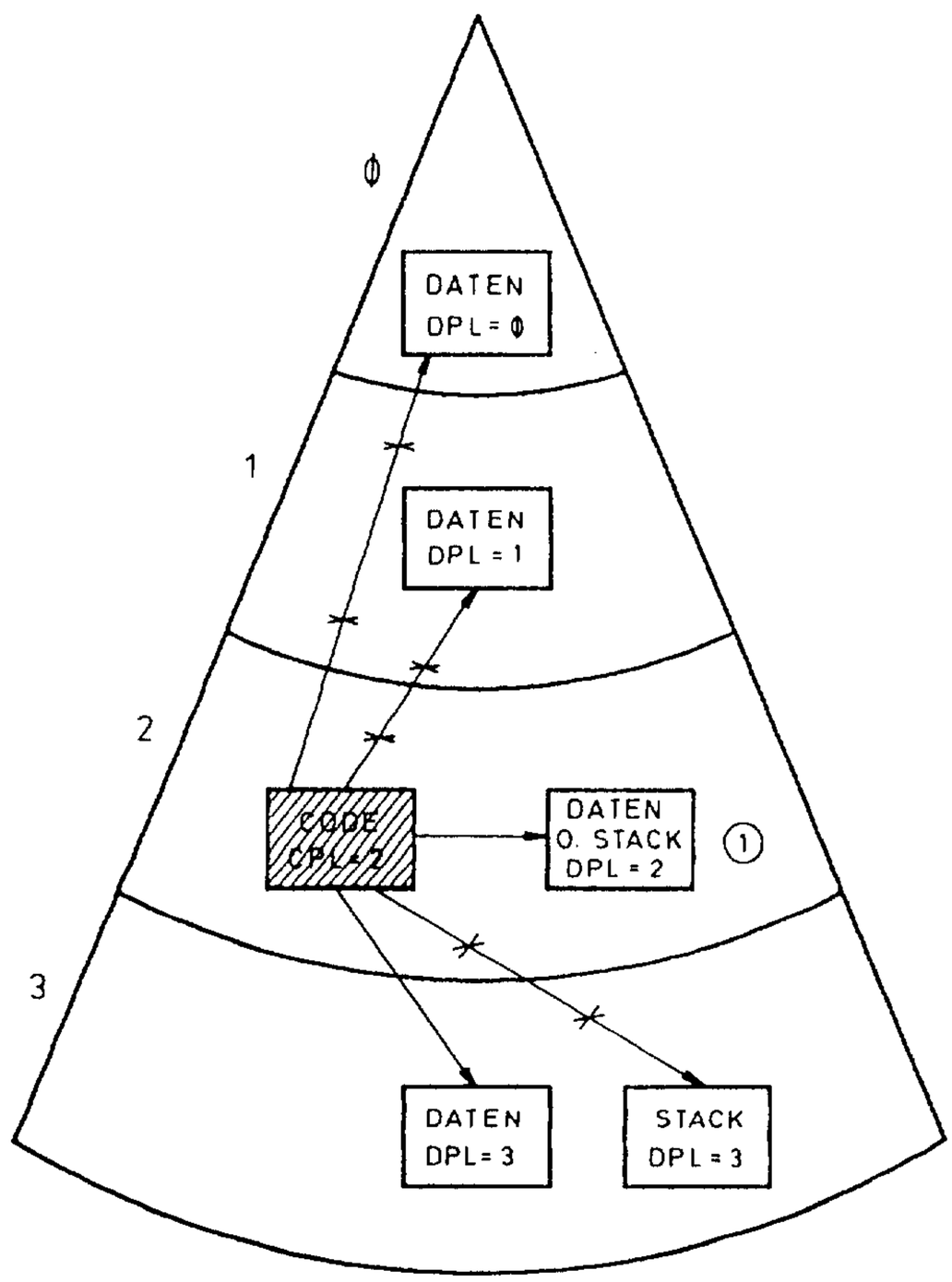
So ist es z. B. nicht möglich, einen "Executable"-Segment-Deskriptor in ein Daten-Segment-Register (DS, ES oder SS) zu laden.

Welche Segment-Register Zugriffe auf die unterschiedlichen Segment-Typen zulassen, zeigt die folgende Tabelle:

Segment Register	Erlaubte Segment Typen			
	Read ***O***nly (RO) Daten Segment	***R***ead-***W***rite (RW) Daten Segment	***E***xecute ***O***nly (EO) Code Segment	***E***xecute-***R***ead (ER) Code Segment
DS	Ja	Ja	Nein	Ja
ES	Ja	Ja	Nein	Ja
SS	Nein	Ja	Nein	Nein
CS	Nein	Nein	Ja	Ja

Es ist ersichtlich, daß z. B. die Segment-Register DS, ES oder SS ***keine*** Zugriffe auf "Execute Only"-Segmente (Typ EO) erlauben.

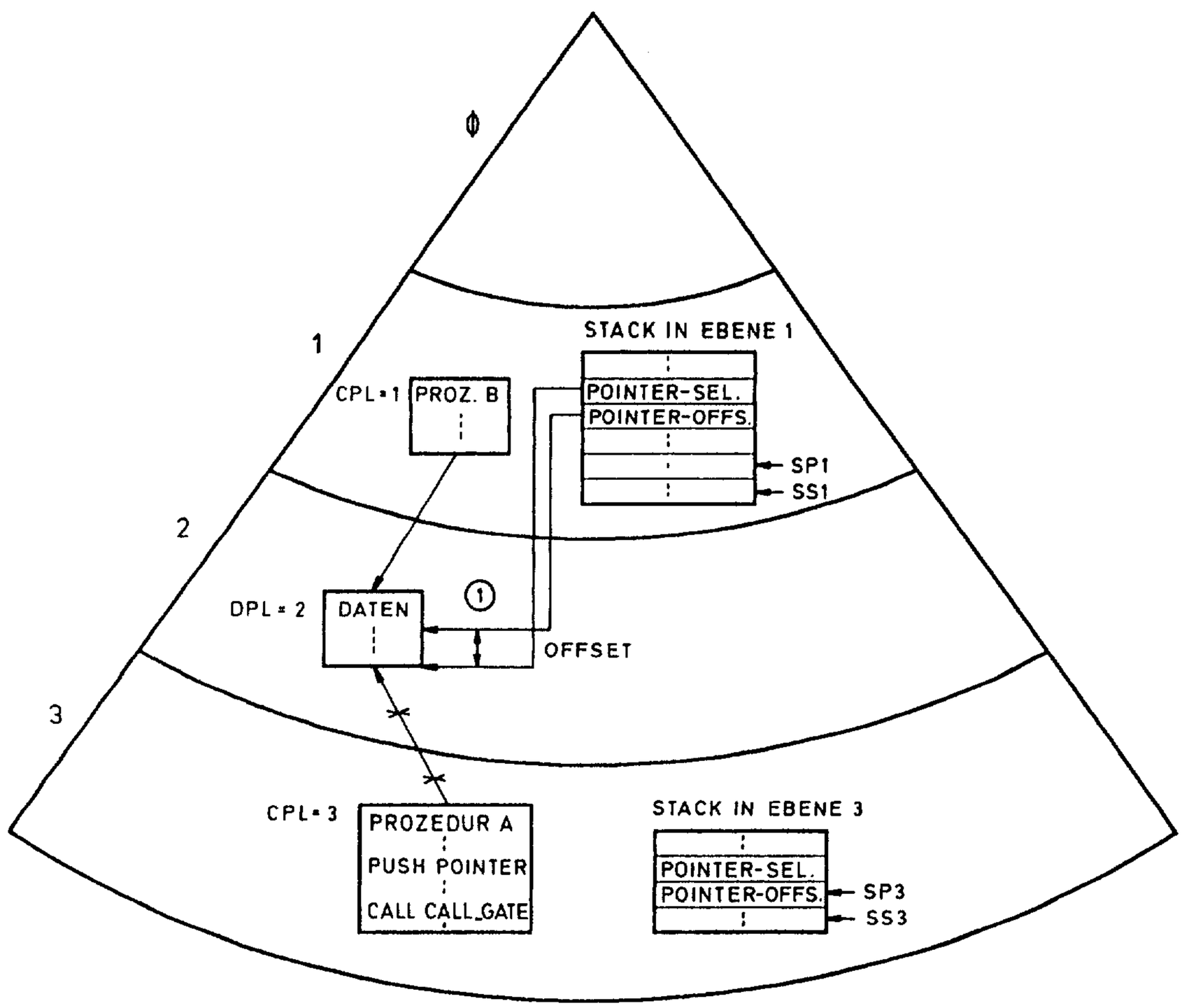
Neben den oben genannten Überprüfungen wird beim Laden eines Segment-Registers auch eine Privileg-Analyse durchgeführt.



Wie das obige Bild zeigt, ist einem Code-Segment der Zugriff auf ein Daten-Segment nur dann erlaubt, wenn DPL des Daten-Segments numerisch ***größer*** oder ***gleich*** CPL der "Task" ist (DPL $\geqslant$ CPL). Lediglich beim Zugriff auf Stack-Daten muß DPL des Stack-Daten-Segments ***gleich*** CPL sein (DPL = CPL) ①.

3.15 Parameter-Gültigkeit

Es gibt einen Typ der Privileg-Verletzung, den die 80286-CPU nicht automatisch erkennen kann. Dies soll am folgenden Beispiel erläutert werden.



Es soll die Prozedur A in der Privilegebene 3 existieren, die vor dem Aufruf der Prozedur B (CALL CALL_GATE) in der Privilegebene 1 einen 32-Bit-Pointer-Parameter (PUSH POINTER) an den Stack der Prozedur B übergibt. Dabei soll Prozedur A das RPL-Feld der Parameter-Selektorkomponente so manipuliert haben, daß RPL ***nicht*** mehr die Privilegebene des aufrufenden Code-Segments, sondern eine höhere Ebene spezifiziert. Im Beispiel soll der Parameter-Selektor den RPL-Wert 2 bekommen. Er ist damit mit der Privilegebene eines Datensegments identisch (DPL = 2), auf das die Prozedur B ein uneingeschränktes Zugriffsrecht hat (DPL $\geqslant$ CPL).

Während die Parameter-Selektorkomponente einen Deskriptor auswählt, der das Datensegment beschreibt, kann nun die Parameter-Offsetkomponente benützt werden, um z. B. eine Datenstruktur in diesem Segment zu manipulieren ①.

Eine solche Situation, ob absichtlich oder unabsichtlich hervorgerufen, würde normalerweise das Schutzkonzept des 80286 verletzen, da die Privilegebene von Prozedur A (CPL = 3) für einen Zugriff auf eine Datenstruktur in Ebene 2 (DPL = 2) ***nicht*** groß genug ist. Allerdings kann die CPU diesen Typ von Privileg-Verletzung nicht erkennen, da die Privilegebene von Prozedur B (CPL = 1), die eigentlich die Datenstruktur adressiert, ***genügend*** groß ist.

Der beschriebene Trick ist deshalb vergleichbar mit der historischen Geschichte vom "Trojanischen Pferd", mit dessen Hilfe griechische Soldaten in die Stadt Troja eindringen konnten. Zur Lösung dieses sogenannten Trojanischen Problems stellt der 80286 jedoch einige Befehle zur Verfügung, die solche Schutzverletzungen überwachen. Vor der Erläuterung dieser Befehle soll zunächst die beschriebene Situation durch das folgende ASM286-Programm realisiert werden.

3.15.1 **Programmierbeispiel:** Privileg-Verletzung

```
iAPX286 MACRO ASSEMBLER    B12

PC 16-20 - iAPX286 MACRO ASSEMBLER VX.X ASSEMBLY OF MODULE B12
OBJECT MODULE PLACED IN B12.OBJ
ASSEMBLER INVOKED BY:  ASM286.86 B12.SRC


LOC  OBJ                  LINE     SOURCE

                            1              NAME    B12
                            2
                            3     ASSUME DS:EBENE_3_DATEN,ES:EBENE_2_EXTRA_DATEN
                            4
                            5     EXTRN CALL_GATE:FAR
                            6
----                        7     EBENE_1_CODE    SEGMENT  EO            ;Nur ausfuehrbares Segment mit
                            8                                            ;Basisadresse=1000H und DPL=1
                            9                                            ;(Festlegung durch BLD286)
----                       10     STACK_FRAME     STRUC
0000                       11     EBENE_3_BP          DW  ?             ;Ebene 3-BP
0002                       12     IP3                 DW  ?             ;Ruecksprung-IP
0004                       13     CS3                 DW  ?             ;Ruecksprung-CS
0006                       14     POINTER_OFFSET      DW  ?             ;Offset des Pointer-Parameters
0008                       15     POINTER_SELEKTOR    DW  ?             ;Selektor des Pointer-Parameters
000A                       16     SP3                 DW  ?             ;Ebene 3-SP
000C                       17     SS3                 DW  ?             ;Ebene 3-SS
----                       18     STACK_FRAME     ENDS
                           19
0000                       20     B   PROC    FAR  WC(2)                ;32-Bit Pointer (2 Worte) an den
                           21                                           ;Ebene 1-Stack uebergeben
0000 55                    22         PUSH    BP                        ;Rette BP des Ebene 3-Stack
0001 8BEC                  23         MOV     BP,SP                     ;Stelle BP an die Spitze des
                           24                                           ;Ebene 1-Stack
                           25         ;
```

```
                        26          ;
0003 8B5E06             27          MOV BX,[BP].POINTER_OFFSET          ;(BX) = Pointer-Offset
0006 8E4608             28          MOV ES,[BP].POINTER_SELEKTOR        ;(ES) = Pointer-Selektor
0009 268A07             29          MOV AL,ES:BYTE PTR[BX]              ;(AL) = 1.tes Element des Vektors
                        30          ;
                        31          ;
000C 5D                 32          POP BP                              ;Lade BP mit dem alten Ebene 3-Wert
000D CA0400             33          RET 4                               ;Entferne 32-Bit Pointer aus dem
                        34                                              ;Ebene 1-Stack
                        35      B   ENDP
                        36
----                    37      EBENE_1_CODE    ENDS
                        38
----                    39      EBENE_2_EXTRA_DATEN SEGMENT RW          ;Les- und schreibbares Segment mit
                        40                                              ;Basis-Adresse 2000H und DPL=2
                        41                                              ;(Festlegung durch BLD286)
0000 00                 42      VEKTOR  DB  0,1,2,3,4,5,6,7,8,9         ;Vektor mit 10 Byte-Elementen
0001 01
0002 02
0003 03
0004 04
0005 05
0006 06
0007 07
0008 08
0009 09
                        43
----                    44      EBENE_2_EXTRA_DATEN ENDS
                        45
----                    46      EBENE_3_DATEN   SEGMENT  RW             ;Les- und schreibbares Segment mit
                        47                                              ;Basisadresse 3000H und DPL=3
                        48                                              ;(Festlegung durch BLD286)
0000 ????????           49      POINTER     DD  ?                       ;An die Prozedur B zu uebergebender
                        50                                              ;32-Bit Pointer-Parameter
                        51
----                    52      EBENE_3_DATEN   ENDS
                        53
----                    54      EBENE_3_CODE    SEGMENT  EO             ;Nur ausfuehrbares Segment mit
                        55                                              ;Basisadresse 4000H und DPL=3
                        56                                              ;(Festlegung durch BLD286)
0000                    57      A   PROC    NEAR
                        58          ;
                        59          ;
0000 C70600000000       60          MOV WORD PTR POINTER,OFFSET VEKTOR  ;POINTER-Offset=Anfangsadresse von
                        61                                              ;                VEKTOR
0006 C70602001A00       62          MOV WORD PTR POINTER+2,0000000000011010B
                        63                                              ;POINTER-Selektor mit
                        64                                              ;        RPL=2
                        65                                              ;        TI=0 (GDT)
                        66                                              ;        Index=3 (Offset=18H)
000C FF360200           67          PUSH    WORD PTR POINTER+2          ;Uebergabe des POINTER-Selektors und
0010 FF360000           68          PUSH    WORD PTR POINTER            ;des POINTER-Offsets an
                        69                                              ;das Stack-Segment in Ebene 1
0014 9A0000----     E   70          CALL    CALL_GATE                   ;Aufruf der Prozedur B in der
                        71                                              ;Privileg-Ebene 1 ueber einen
                        72                                              ;Call Gate-Deskriptor
                        73                                              ;(Festlegung durch BLD286)
                        74          ;
                        75          ;
```

```
                          76      A   ENDP
                          77
....                      78      EBENE_3_CODE ENDS
                          79
                          80          END

ASSEMBLY COMPLETE,   NO WARNINGS,   NO ERRORS
```

Das angegebene Programm-Fragment enthält insgesamt ***vier*** Segmente:

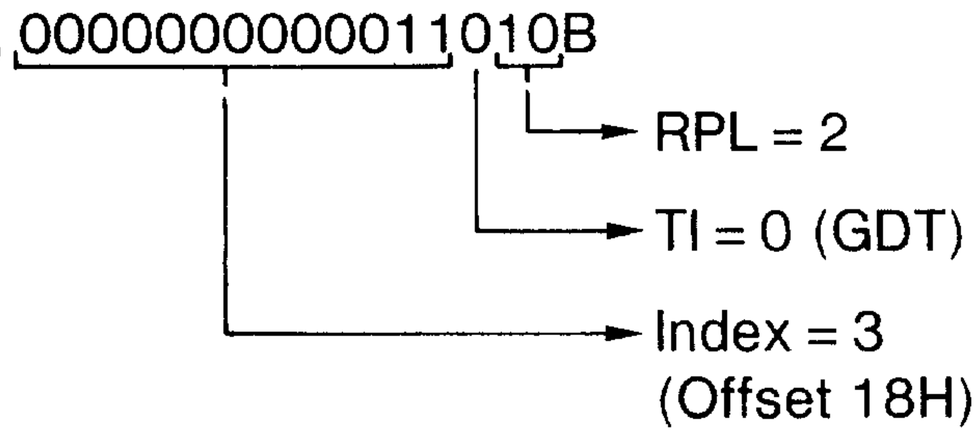
- EBENE_1_CODE in der Privilegebene 1
- EBENE_2_EXTRA_DATEN in der Privilegebene 2
- EBENE_3_DATEN in der Privilegebene 3
- EBENE_3_CODE in der Privilegebene 3

Das EBENE_3_CODE-Segment enthält die Prozedur A, in der zunächst ein 32-Bit-Pointer zur Datenstruktur VEKTOR im EBENE_2_EXTRA_DATEN-Segment generiert wird. Dabei benützt diese Prozedur die "Double Word"-Variable POINTER im EBENE_3_DATEN-Segment, um darin mit

- MOV WORD PTR POINTER, OFFSET VEKTOR

den ***Offset*** von VEKTOR, und mit

- MOV WORD PTR POINTER+2, 0000000000011010B



den Selektor des EBENE_2_EXTRA_DATEN-Segments zu speichern. Das RPL-Feld des Selektors ist mit dem Wert 2 initialisiert. Dies bedeutet, daß der Selektor höher privilegiert ist, als das EBENE_3_CODE-Segment, das diesen Selektor liefert.

Den in der POINTER-Variablen hinterlegten 32-Bit-Pointer speichert die Prozedur A mit

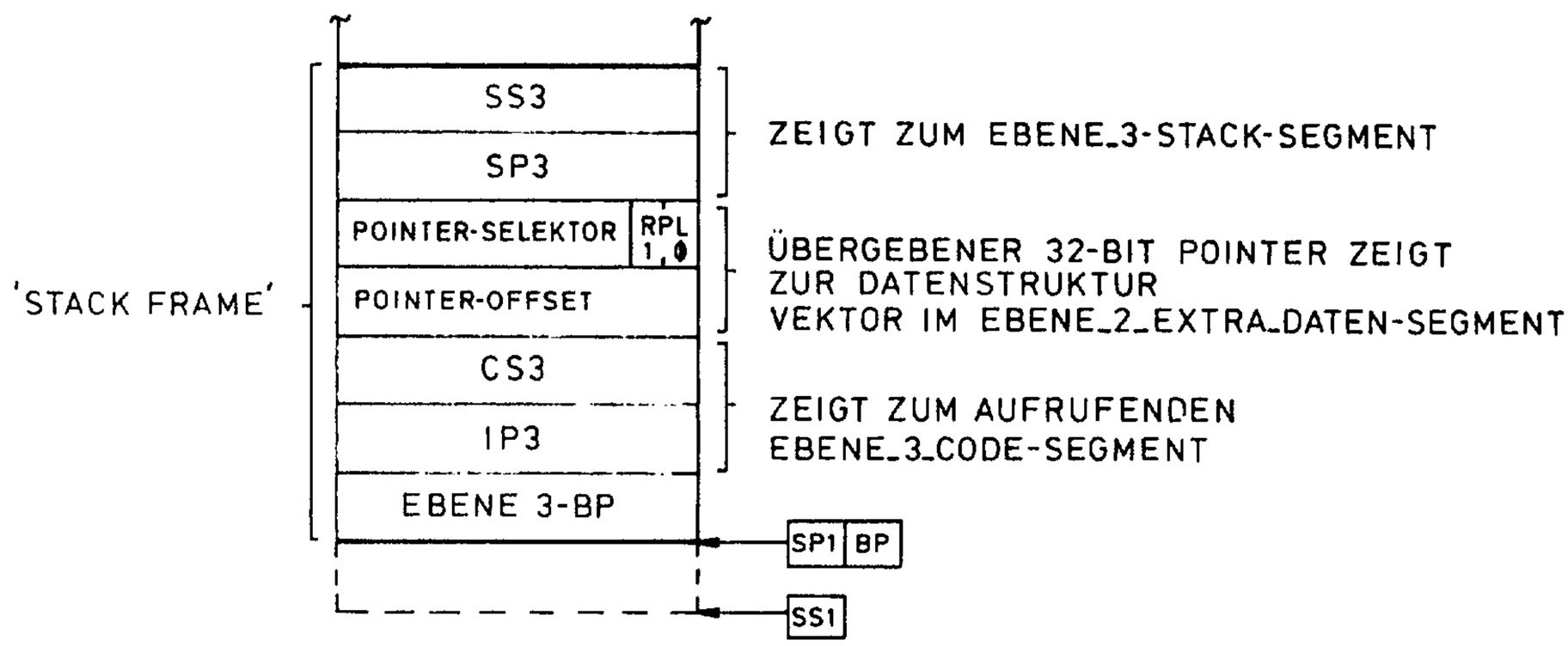
- PUSH WORD PTR POINTER+2
- PUSH WORD PTR POINTER

im Ebene-3-Stack und ruft danach mit

- CALL CALL_GATE

über einen "Call Gate"-Deskriptor, die FAR-Prozedur B im EBENE_1_CODE-Segment in der Privilegebene 1 auf.

Das Ebene 1-Stack-Layout nach Aufruf der Prozedur B zeigt das folgende Bild:



Damit der Zugriff auf den 32-Bit-Pointer in der Prozedur B im EBENE_1_CODE-Segment bequem programmiert werden kann, ist das Ebene-1-Stack-Layout als ASM286-Struktur mit dem Namen STACK_FRAME vereinbart.

```
        STACK_FRAME          STRUC
[BP]--> EBENE_3_BP           DW  ?  ; Ebene 3-BP
        IP3                  DW  ?  ; Rücksprung-IP
        CS3                  DW  ?  ; Rücksprung-CS
        POINTER_OFFSET       DW  ?  ; Offset des Pointer-Parameters
        POINTER_SELECTOR     DW  ?  ; Selektor des Pointer-Parameters
        SP3                  DW  ?  ; Ebene 3-SP
        SS3                  DW  ?  ; Ebene 3-SS
        STACK_FRAME          ENDS
```

Die Übernahme des Pointers aus dem Ebene-1-Stack geschieht dann mit

- MOV BX,[BP]. POINTER_OFFSET
- MOV ES, [BP]. POINTER_SELEKTOR

Dadurch wird BX mit dem Offset von VEKTOR und ES mit dem Selektor des EBENE_2_EXTRA_DATEN-Segments geladen.

Dem Pointer-Selektor mit dem manipulierten RPL-Wert 2 ist es erlaubt, den Deskriptor des EBENE_2_EXTRA-DATEN-Segments mit DPL = 2 in das ES-Cache-Register zu laden (RPL ⩽ DPL). Da sich das EBENE_1_CODE-Segment in der Privilegebene 1 befindet (CPL = 1), kann nun die darin enthaltene Prozedur B uneingeschränkt auf die Datenstruktur VEKTOR im EBENE_2_EXTRA-DATEN-Segment zugreifen (DPL ⩾ CPL). Ein solcher Zugriff ist im Programm-Fragment durch

- MOV AL,ES:BYTE PTR [BX]

illustriert.

3.15.2 Anwendung des ARPL-Befehls

Um das Problem des unberechtigten Segment-Zugriffs durch beabsichtigte oder unbeabsichtigte Privileg-Verletzung zu vermeiden, müssen höher privilegierte Code-Segmente die Möglichkeit haben, die Privilegebene des übergebenen Selektors auf das niedrigere Niveau des aufrufenden Code-Segments zu reduzieren.

Wie das obige Beispiel zeigte, sind Zugriffe auf das EBENE_2_EXTRA_DATEN-Segment nur deswegen möglich, weil die Bedingung Pointer-Selektor RPL ⩽ Deskriptor DPL (2 = 2) erfüllt ist. Wenn das RPL-Feld des Pointer-Selektors mit der Privilegebenen-Kennziffer des aufrufenden Code-Segments initialisiert werden könnte (im Beispiel mit dem Wert 3), wäre die oben genannte Bedingung ***nicht*** mehr erfüllt. Benützt das aufrufende Code-Segment diesen ***angepaßten*** Selektor, stellt dies eine Ausnahmesituation dar und führt zur Unterbrechung des Programms.

Um eine solche Privileg-Anpassung des übergebenen Pointer-Selektors zu realisieren, stellt die 80286-CPU den Befehl

- ARPL SELEKTOR_A,SELEKTOR_B (Adjust RPL Field of Selector)

zur Verfügung.

Dieser Befehl hat zwei Operanden:

- Der erste Operand (SELEKTOR_A) ist eine 16-Bit-Speichervariable oder ein Wort-Register und enthält einen Selektor, dessen RPL-Feld angepaßt werden soll.
- Der zweite Operand (SELEKTOR_B) ist ein Wort-Register und enthält ebenfalls einen Selektor, dessen RPL-Feld den Anpassungswert liefert.

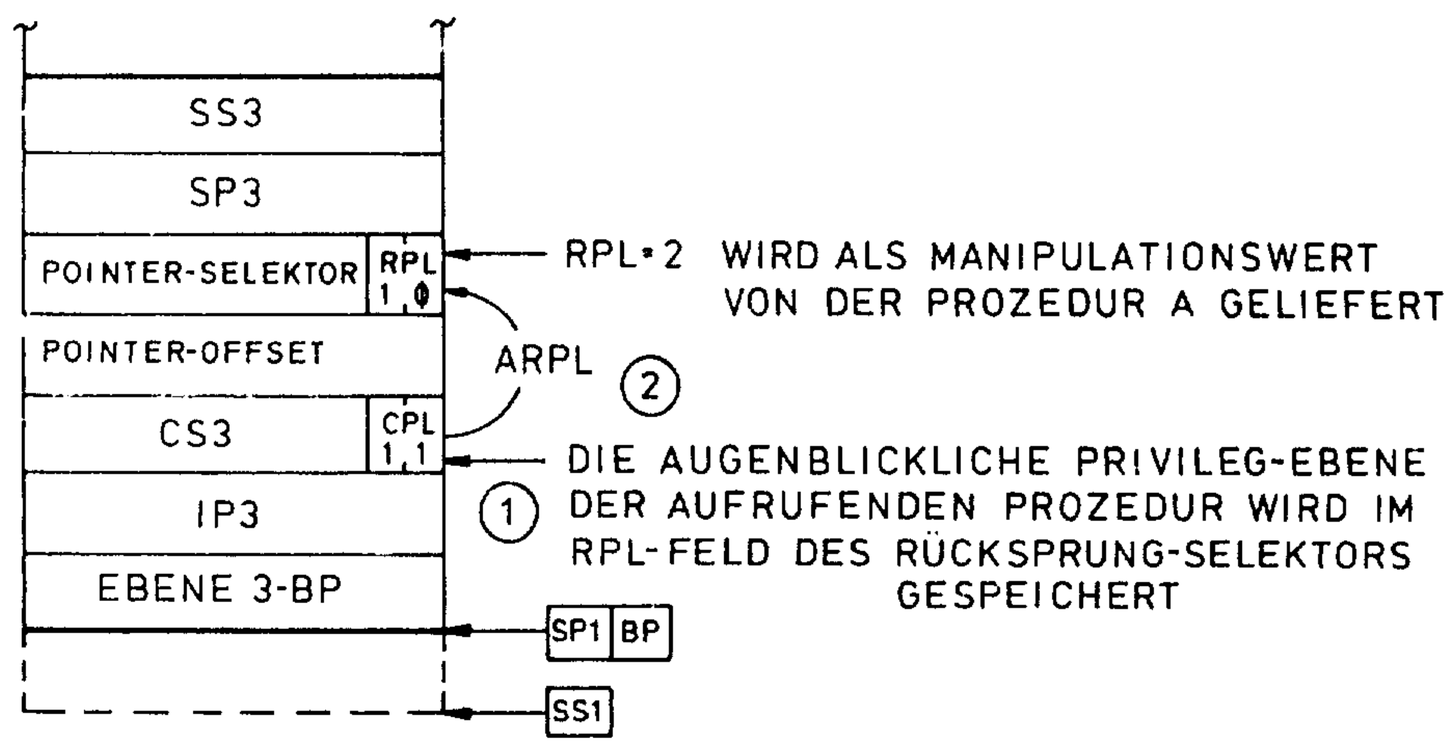
Wenn das RPL-Feld des ersten Operanden ***kleiner*** ist als das RPL-Feld des zweiten Operanden, wird das

- "Zero Flag" auf 1 gesetzt, und
- das RPL-Feld des ersten Operanden bekommt den RPL-Wert des zweiten Operanden.

Wenn das RPL-Feld des ersten Operanden ***größer*** ist als das RPL-Feld des zweiten Operanden, wird das

- "Zero Flag" auf 0 gesetzt, und
- das RPL-Feld des ersten Operanden behält seinen Wert.

Wie der ARPL-Befehl zu benützen ist, soll anhand des Programmierbeispiels einer Privileg-Verletzung erläutert werden. Wie bereits gezeigt worden ist, ergibt sich nach dem Aufruf der Prozedur B das folgende Ebene-1-Stack-Layout:



Es ist ersichtlich, daß die augenblickliche Privilegebene (CPL = 3) der aufrufenden Prozedur A (im EBENE__3__CODE-Segment) im RPL-Feld des Rücksprung-Selektors CS3 gespeichert ist ①.
Diesen Wert kann ARPL benützen, um damit das RPL-Feld des Pointer-Selektors zu überschreiben ②.

Um den Rücksprung-Selektor CS3 und den Pointer-Selektor im Ebene-1-Stack zu erreichen, kann die bereits definierte Struktur STACK_FRAME benützt werden:

```
        STACK_FRAME          STRUC
[BP]---->EBENE_3_BP          DW  ?   ; Ebene 3-BP
         IP3                 DW  ?   ; Rücksprung-IP
         CS3                 DW  ?   ; Rücksprung-CS
         POINTER_OFFSET      DW  ?   ; Offset des Pointer-Parameters
         POINTER_SELEKTOR    DW  ?   ; Selektor des Pointer-Parameters
         SP3                 DW  ?   ; Ebene 3-SP
         SS3                 DW  ?   ; Ebene 3-SS
         STACK_FRAME         ENDS
```

Dabei ist mit [BP].CS3 der Rücksprung-Selektor CS3 mit CPL = 3 und mit [BP].POINTER_SELEKTOR der Pointer-Selektor mit RPL = 2 zu erreichen.

Um nun sicher zu gehen, daß RPL des Pointer-Selektors numerisch ***nicht*** kleiner ist als CPL der aufrufenden Prozedur, kann die folgende Form von Anweisungen benützt werden:

```
● MOV AX,[BP].CS3                  ; (AX) = Rücksprung-Selektor
● ARPL [BP].POINTER_SELEKTOR,AX
                                   ; RPL-Feld des Rückprung-
                                   ; Selektors liefert den
                                   ; Anpassungswert 3
                                   ; RPL-Feld des Pointer-
                                   ; Selektors bekommt den
                                   ; Anpassungswert 3
```

Es sei darauf hingewiesen, daß ARPL in der Betriebssystem-Software erscheint und ***nicht*** in Anwenderprogrammen. Außerdem wird ARPL im "Real Address Mode" ***nicht*** erkannt. Er sollte von jeder Betriebssystem-Prozedur verwendet werden, zumal er einen "preiswerten" Schutz gegen eine versehentliche Benützung ungültiger Parameter bietet.

3.15.3 **Programmierbeispiel:** Privileg-Anpassung

Das folgende Programm-Fragment entspricht dem Programmierbeispiel "Privileg-Verletzung" und soll zeigen, wie eine privilegierte Datenstruktur durch den ARPL-Befehl gegen den Zugriff eines ungültigen Pointer-Parameters geschützt werden kann.

```
iAPX286 MACRO ASSEMBLER    B13


PC 16-20 - iAPX286 MACRO ASSEMBLER VX.X ASSEMBLY OF MODULE B13
OBJECT MODULE PLACED IN B13.OBJ
ASSEMBLER INVOKED BY:  ASM286.86 B13.SRC


LOC  OBJ                  LINE     SOURCE

                            1              NAME    B13
                            2
                            3     ASSUME DS:EBENE_3_DATEN,ES:EBENE_2_EXTRA_DATEN
                            4
                            5     EXTRN CALL_GATE:FAR
                            6
----                        7     EBENE_1_CODE    SEGMENT  EO            ;Nur ausfuehrbares Segment mit
                            8                                            ;Basisadresse=1000H und DPL=1
                            9                                            ;(Festlegung durch BLD286)
----                       10     STACK_FRAME     STRUC
0000                       11     EBENE_3_BP          DW  ?             ;Ebene 3-BP
0002                       12     IP3                 DW  ?             ;Ruecksprung-IP
0004                       13     CS3                 DW  ?             ;Ruecksprung-CS
0006                       14     POINTER_OFFSET      DW  ?             ;Offset des Pointer-Parameters
0008                       15     POINTER_SELEKTOR    DW  ?             ;Selektor des Pointer-Parameters
000A                       16     SP3                 DW  ?             ;Ebene 3-SP
000C                       17     SS3                 DW  ?             ;Ebene 3-SS
----                       18     STACK_FRAME     ENDS
                           19
0000                       20     B   PROC    FAR  WC(2)                ;32-Bit Pointer (2 Worte) an den
                           21                                           ;Ebene 1-Stack uebergeben
0000 55                    22         PUSH    BP                        ;Rette BP des Ebene 3-Stack
0001 8BEC                  23         MOV     BP,SP                     ;Stelle BP an die Spitze des
                           24                                           ;Ebene 1-Stack
0003 884604                25         MOV AX,[BP].CS3                   ;(AX)=Ruecksprung-Selektor
                           26                                           ;CPL-Feld liefert Anpassungswert
0006 634608                27         ARPL    [BP].POINTER_SELEKTOR,AX  ;RPL-Feld des Pointer-Selektor bekommt
                           28                                           ;den Anpassungswert
0009 7409                  29         JZ  RPL_ANPASSUNG                 ;Sprung, wenn RPL des Pointer-Selektors
                           30                                           ;angepasst worden ist (Zero Flag=1)
                           31                                           ;Erfolgte keine Anpassung, kann
                           32                                           ;Prozedur B auf den Vektor zugreifen
000B 8B5E06                33         MOV BX,[BP].POINTER_OFFSET        ;(BX) = Pointer-Offset
000E 8E4608                34         MOV ES,[BP].POINTER_SELEKTOR      ;(ES) = Pointer-Selektor
0011 268A07                35         MOV AL,ES:BYTE PTR[BX]            ;(AL) = 1.tes Element von VEKTOR
                           36         ;
                           37         ;
0014                       38     RPL_ANPASSUNG:                        ;Prozedur A kann Prozedur B nicht fuer
                           39                                           ;einen Zugriff auf den VEKTOR
```

```
                                                    ;benuetzen, daher Prozedur B verlassen
0014 5D                 POP BP                      ;Lade BP mit dem alten Ebene 3-Wert
0015 CA0400             RET 4                       ;Entferne 32-Bit Pointer aus dem
                                                    ;Ebene 1-Stack
                    B   ENDP

....                EBENE_1_CODE    ENDS

....                EBENE_2_EXTRA_DATEN SEGMENT RW  ;Les- und schreibbares Segment mit
                                                    ;Basis-Adresse 2000H und DPL=2
                                                    ;(Festlegung durch BLD286)
0000 (10            VEKTOR          DB  10 DUP(?)   ;Vektor mit 10 Byte-Elementen
     ??
     )

....                EBENE_2_EXTRA_DATEN ENDS

....                EBENE_3_DATEN   SEGMENT  RW     ;Les- und schreibbares Segment mit
                                                    ;Basisadresse 3000H und DPL=3
                                                    ;(Festlegung durch BLD286)
0000 ????????       POINTER         DD  ?           ;An die Prozedur B zu uebergebender
                                                    ;32-Bit Pointer-Parameter
0004 (10            EBENE_2_DATEN   DB  10 DUP(?)   ;Von VEKTOR uebernommene Elemente
     ??
     )

....                EBENE_3_DATEN   ENDS

....                EBENE_3_CODE    SEGMENT  EO     ;Nur ausfuehrbares Segment mit
                                                    ;Basisadresse 4000H und DPL=3
                                                    ;(Festlegung durch BLD286)
0000                A   PROC    NEAR
                        ;
                        ;
0000 C70600000000       MOV WORD PTR POINTER,OFFSET VEKTOR  ;POINTER-Offset=Anfangsadresse von
                                                    ;                VEKTOR
0006 C70602001A00       MOV WORD PTR POINTER+2,0000000000011010B
                                                    ;POINTER-Selektor mit
                                                    ;        RPL=2
                                                    ;        TI=0 (GDT)
                                                    ;        Index=3 (Offset=18H)
000C FF360200           PUSH    WORD PTR POINTER+2  ;Uebergabe des POINTER-Selektors und
0010 FF360000           PUSH    WORD PTR POINTER    ;des POINTER-Offsets an
                                                    ;das Stack-Segment in Ebene 1
0014 9A0000----     E   CALL    CALL_GATE           ;Aufruf der Prozedur B in der
                                                    ;Privileg-Ebene 1 ueber einen
                                                    ;Call Gate-Deskriptor
                                                    ;(Festlegung durch BLD286)
0019 A20400             MOV     EBENE_2_DATEN,AL    ;(AL) = 1.tes Element von VEKTOR an
                                                    ;       EBENE_2_DATEN uebergeben
                        ;
                        ;
                    A   ENDP

....                EBENE_3_CODE ENDS

                        END

ASSEMBLY COMPLETE,   NO WARNINGS,   NO ERRORS
```

Aus dem Programm-Listing ist ersichtlich, daß in der Prozedur B vor der Übernahme des POINTER-Parameters aus dem Ebene-1-Stack mit

- MOV AX, [BP].CS3
- ARPL [BP].POINTER_SELEKTOR,AX

zunächst überprüft wird, ob RPL des POINTER_SELEKTOR's an CPL des Rücksprung-Selektors CS3 angepaßt werden muß. Da RPL(2) < CPL(3) ist, führt ARPL die Anpassung durch und setzt das "Zero Flag" = 1.

Da jetzt ein Zugriff auf den VEKTOR im EBENE_2_EXTRA_DATEN-Segment ***nicht*** erlaubt ist, wird der Befehl

- JZ RPL_ANPASSUNG

ausgeführt. Er bewirkt, daß die Prozedur B durch

- POP BP
- RET 4

verlassen wird.

Wäre die Anpassung ***nicht*** vorgenommen worden, hätte Prozedur B den POINTER-Parameter geladen und anschließend mit

- MOV AL, ES:BYTE PTR [BX]

das erste Element von VEKTOR im AL-Register zwischengespeichert.

Nach dem Verlassen der Prozedur B würde nun die Prozedur A das AL-Register benützen, um mit

- MOV EBENE_2_DATEN, AL

das vom EBENE_2_EXTRA_DATEN-Segment (Privilegebene 2) geholte Element in das EBENE_3_DATEN-Segment (Privilegebene 3) zu übernehmen (Trojanisches Problem).

3.16 Genaue Überprüfung von Pointer-Parametern

Während der ARPL-Befehl alleine ausreicht, um ***ungültige*** Parameter zu erkennen, kann trotzdem eine Ausnahmesituation entstehen, wenn eine Prozedur einen Parameter benützt. So könnte es z. B. vorkommen, daß die Prozedur ein Segment ***lesen*** will, obwohl der Parameter-Selektor einen Deskriptor geladen hat, der dieses Segment als ***nur ausführbar*** spezifiziert.

Da normalerweise nach dem Aufruf der Prozedur mehrere Befehle ausgeführt werden, kann eine solche Ausnahmesituation erst dann auftreten, wenn die aufgerufene Prozedur den Parameter bekommt.

Um Parameterfehler ***frühzeitig*** zu erkennen, sollte das Betriebssystem die Pointer-Parameter mit den Befehlen

- VERR (Verify a Segment for Reading)
- VERW (Verify a Segment for Writing)
- LAR (Load Access Rights Byte)
- LSL (Load Segment Limit)

überprüfen.

Als Strategie zur genauen Überprüfung der Gültigkeit von Pointer-Parametern können folgende Aktionen vorgeschlagen werden:

- Benütze ARPL, um sicher zu sein, daß das RPL-Feld des Pointer-Parameters die Privilegebene der aufrufenden Prozedur enthält.
- Benütze VERR oder VERW, um sicher zu sein, daß das gekennzeichnete Segment in der Privilegebene der aufrufenden Prozedur erreichbar ist. VERR bestimmt auch, ob das gekennzeichnete Segment ***lesbar*** ist, da z. B. ein ***nur ausführbares*** Segment nicht gelesen werden kann. Daneben bestimmt VERW, ob das Segment ***beschreibbar*** ist.
- Benütze LAR und LSL, um sicher zu sein, daß die Offset-Komponente auf eine Position innerhalb der Segmentgrenzen zeigt. Dabei liefert LAR das "Access"-Byte des aktuellen Deskriptors, so daß bestimmt werden kann, ob das Segment z. B. ein "Expand Down"-Datensegment ist.

LSL liefert das Segment-Limit-Feld des aktuellen Deskriptors. Wenn das Segment ein "Expand-Down"-Datensegment ist, muß die Offset-Komponente ***größer*** als die Segment-Untergrenze sein. Im anderen Fall muß der Offset ***kleiner*** oder ***gleich*** der Segment-Obergrenze sein.

Die angegebene Strategie zur Überprüfung der Gültigkeit von Pointer-Parametern ist "kostspieliger" als die alleinige Benützung des ARPL-Befehls. Dabei ist es vernünftig, die Anwendung dieser Strategie auf solche Betriebssystem-Prozeduren zu beschränken, die von ***weniger*** privilegierten Anwender-Prozeduren aufgerufen werden können.

3.16.1 Anwendung der Befehle VERR und VERW

- `VERR ew` ; Verify a Segment for Reading
- `VERW ew` ; Verify a Segment for Writing

Der in beiden Befehlen angegebene Operand ew ist ein Wort-Register oder ein Word-Speicheroperand, der den Wert eines Selektors enthält.
Die Befehle bestimmen, ob das durch den Selektor gekennzeichnete Segment von der augenblicklichen Privilegebene erreichbar ist und ob das Segment gelesen oder beschrieben werden kann.

- Wenn auf das Segment ***zugegriffen*** werden kann, wird das "Zero Flag" auf 1 gesetzt.
- Wenn auf das Segment ***nicht zugegriffen*** werden kann, wird das "Zero Flag" auf 0 gesetzt.

Damit das "Zero Flag" ***gesetzt*** werden kann, müssen noch folgende Bedingungen erfüllt sein:

- Der Selektor muß den Deskriptor eines ***Code***- oder ***Daten***-Segments kennzeichnen, wobei sich der Deskriptor in der ***GDT*** oder ***LDT*** aufhalten muß.
- Wenn der Befehl VERR benützt wird, muß das Segment ***lesbar*** sein. Dies bedeutet, daß die Zugriffsrechte des Segments durch die Attribute RW (Read-Write) oder ER (Execute-Read) oder RO (Read-Only) gekennzeichnet sein müssen.
- Wenn der Befehl VERW benützt wird, muß das Zugriffsrecht des Segments durch das Attribut RW (Read-Write) gekennzeichnet sein.
- Wenn auf ein Code-Segment zugegriffen wird, das ***lesbar*** und ***nicht "conforming"*** ("non-conforming") ist, muß die Deskriptor-Privilegebene DPL dieses Segments numerisch größer oder gleich der augenblicklichen Privilegebene CPL der aufrufenden Prozedur und RPL des übergebenen Selektors sein (DPL $\geqslant$ CPL, RPL).
- Wenn auf ein Code-Segment zugegriffen wird, das ***lesbar*** und ***"conforming"*** ist, kann die Deskriptor-Privilegebene DPL dieses Segments einen beliebigen Wert haben.

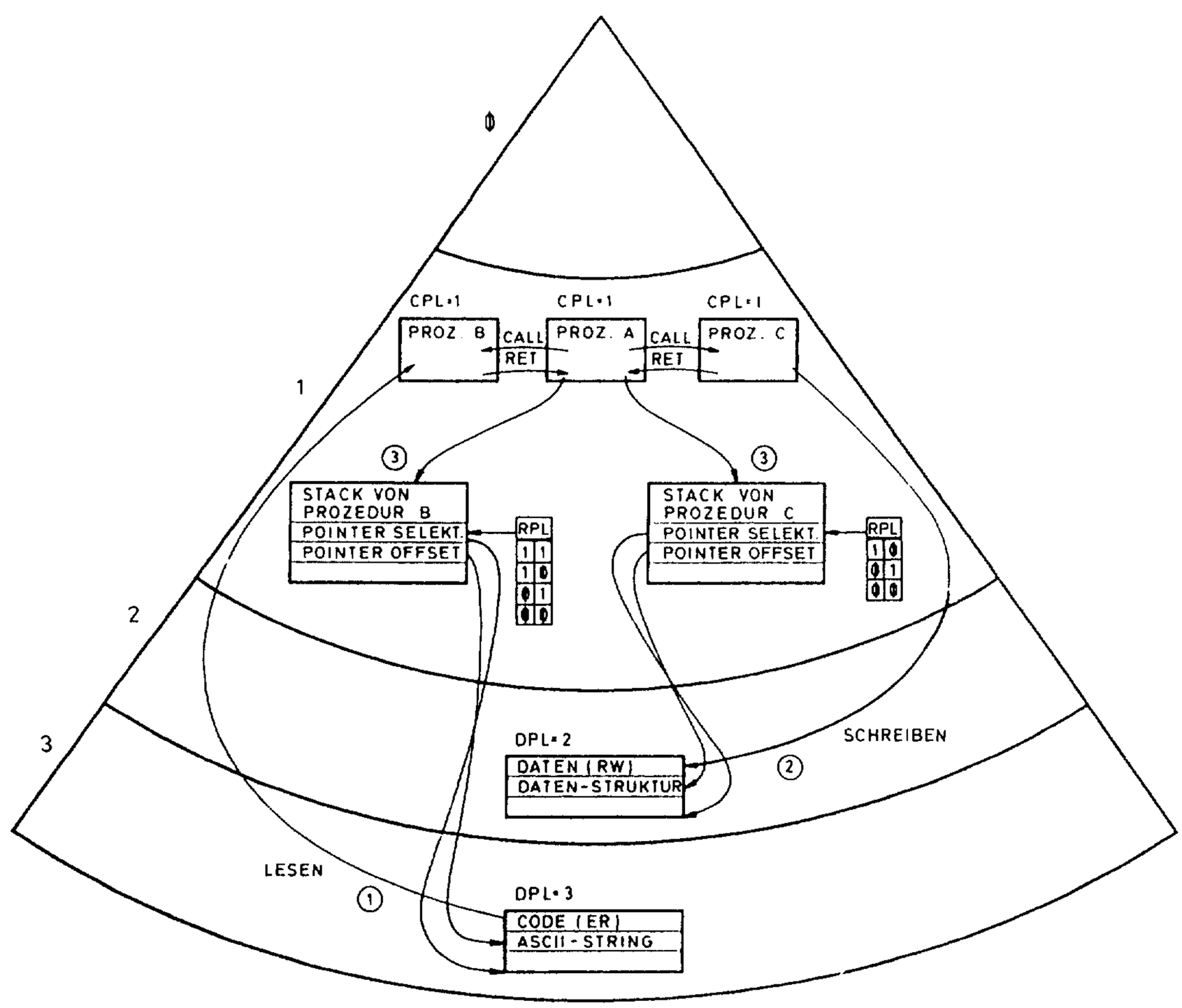
Die von VERR oder VERW geleisteten Gültigkeits-Überprüfungen sind die gleichen, die beim Laden von DS oder ES mit

- MOV AX,SELEKTOR_1
  MOV DS,AX
  oder
- MOV AX,SELEKTOR_2
  MOV ES,AX

durchgeführt werden.
Der einzige Unterschied ist der, daß das Ergebnis der Gültigkeits-Überprüfung im "Zero Flag" angezeigt wird, also zu keiner Ausnahmesituation führt. Dies gibt der Software die Möglichkeit, eventuelle Segment-Zugriffsprobleme zu verhindern.

Es sei darauf hingewiesen, daß eine Programm-Unterbrechung erfolgt, wenn die Befehle VERR und VERW im "Real Address"-Modus ausgeführt werden.

Um zu zeigen, wie die Befehle VERR und VERW benützt werden können, soll eine Situation vorliegen, wie sie das folgende Bild zeigt:



In der Privilegebene 1 existiert die Prozedur B, die einen konstanten ASCII-String in einem Nicht-"conforming"-Code-Segment (Typ ER) in der Privilegebene 3 (DPL = 3) ***liest*** ①.

Daneben existiert in der Privilegebene 1 die Prozedur C, die das Daten-Segment in der Privilegebene 2 (DPL = 2) ***beschreibt*** ②.

Prozedur A benützt die Prozeduren B und C, um entweder auf das Code- oder das Daten-Segment zuzugreifen. Zu diesem Zweck übergibt A an B oder C passende Pointer ③.

Da z. B in dynamischen Systemen die Prozedur A den Typ des Code- bzw. Daten-Segments nicht kennt, wird sie vor dem Aufruf der Prozeduren B oder C zunächst die zu übergebenden Pointer-Selektoren benützen, um die Zugriffsrechte festzustellen.

> **HINWEIS:**
> Dynamische Systeme sind solche, bei denen die Deskriptoren durch das Programm manipuliert werden.

Die Prozedur A wird also überprüfen, ob das Code-Segment lesbar und ob das Daten-Segment beschreibbar ist.

Setzt dabei der Befehl VERR das "Zero Flag" auf 1, kann Prozedur A die Prozedur B aufrufen, um den ASCII-String im Code-Segment zu ***lesen***.

Setzt der Befehl VERW das "Zero Flag" auf 1, kann Prozedur A die Prozedur C aufrufen, um z. B. eine Datenstruktur im Daten-Segment zu beschreiben.

Aus dem obigen Bild ist ersichtlich, daß die Prozeduren A und C den CPL-Wert 1 haben. Damit das "Zero Flag" gesetzt werden kann, muß die Grundbedingung DPL $\geqslant$ (RPL, CPL) erfüllt sein. Daher kann für den Zugriff auf das Code-Segment die Prozedur A Pointer-Selektoren mit RPL = 3, 2, 1 oder 0 übergeben. Für den Zugriff auf das Daten-Segment kann die Prozedur A Pointer-Selektoren mit RPL = 2, 1 oder 0 an die Prozedur C übergeben.

Das folgende Programmierbeispiel – Zugriffs-Überprüfung – enthält den ASM286-Code für die dargestellte Situation.
Es zeigt, daß zunächst in der Prozedur A mit VERR überprüft wird, ob das EBENE_3_ CODE-Segment, das einen ASCII-String enthält, ***gelesen*** werden kann.

Ist dies der Fall, übergibt Prozedur A mit

- PUSH DS ; Selektor des EBENE_3_CO-
  und ; DE-Segments
- PUSH OFFSET ASCII_STRING

einen passenden 32-Bit-Pointer an die Prozedur B, die dann ihrerseits mit

- MOV AL, DS:BYTE PTR [BX]

den ASCII-String liest. Das EBENE_3_CODE-Segment kann deswegen DS-relativ (auch ES-relativ) gelesen werden, weil es vom Typ ER (Execute-Read) ist.

Anschließend überprüft die Prozedur A mit VERW, ob das EBENE_2_DATEN-Segment ***beschreibbar*** ist.
Ist dies der Fall, übergibt Prozedur A mit

- PUSH DS ; Selektor des EBENE_2_DA-
  und ; TEN-Segments
- PUSH OFFSET VEKTOR

einen passenden 32-Bit-Pointer an die Prozedur C, die dann ihrerseits mit

- MOV BYTE PTR [BX], AL

in die Datenstruktur VEKTOR schreibt.

Nur dann, wenn Prozedur B auf das EBENE_3_CODE-Segment ***lesend*** und Prozedur C auf das EBENE_2_DATEN-Segment ***schreibend*** zugreifen kann, wird ein Programm-Pfad durchlaufen, der ein Zeichen von ASCII_STRING nach VEKTOR kopiert.

3.16.2 **Programmierbeispiel:** Zugriffs-Überprüfung

```
iAPX286 MACRO ASSEMBLER    B14

PC 16-20 - iAPX286 MACRO ASSEMBLER VX.X ASSEMBLY OF MODULE B14
OBJECT MODULE PLACED IN B14.OBJ
ASSEMBLER INVOKED BY:  ASM286.86 B14.SRC


LOC  OBJ                  LINE     SOURCE

                             1            NAME    B14
                             2
                             3     ASSUME DS:EBENE_2_DATEN
                             4
....                         5     EBENE_1_CODE    SEGMENT  EO            ;Nur ausfuehrbares Segment mit
                             6                                            ;Basisadresse=1000H und DPL=1
```

```
                                                                  ;(Festlegung durch BLD286)
----          STACK_FRAME     STRUC
0000          PROZEDUR_A_BP       DW  ?                           ;Prozedur A-BP
0002          IP1                 DW  ?                           ;Ruecksprung-IP
0004          CS1                 DW  ?                           ;Ruecksprung-CS
0006          POINTER             DD  ?                           ;32-Bit Pointer-Parameter
----          STACK_FRAME     ENDS

0000          A   PROC    NEAR

0000 B8----       R     MOV     AX,EBENE_3_CODE                   ;(AX)=Selektor von EBENE_3_CODE
0003 8ED8             MOV     DS,AX                             ;Selektor laden
0005 0F00E0           VERR    AX                                ;Ist EBENE_3_CODE-Segment lesbar
0008 7400             JZ      LESBAR                            ;Sprung, wenn lesbar (Zero Flag=1)
                      ;
                      ;
000A          LESBAR:
000A 1E               PUSH    DS                                ;Pointer-Selektor in Ebene 1-Stack
000B 6A0090           PUSH    OFFSET ASCII_STRING               ;Pointer-Offset in Ebene 1-Stack
000E 9A0000----   R   CALL    FAR PTR B                         ;Aufruf von Prozedur B inter-segment
                                                                ;direkt
0013 B9----       R   MOV     CX,EBENE_2_DATEN                  ;(CX)=Selektor von EBENE_2_DATEN
0016 8ED9             MOV     DS,CX                             ;Selektor laden
0018 0F00E9           VERW    CX                                ;Ist EBENE_2_DATEN-Segment beschreibbar
001B 7400             JZ      BESCHREIBBAR                      ;Sprung, wenn beschreibbar(Zero Flag=1)
                      ;
                      ;
001D          BESCHREIBBAR:
001D 1E               PUSH    DS                                ;Pointer-Selektor in Ebene 1-Stack
001E 6A0090           PUSH    OFFSET VEKTOR                     ;Pointer-Offset in Ebene 1-Stack
0021 9A0C00----   R   CALL    FAR PTR C                         ;Aufruf von Prozedur C inter-segment
                                                                ;direkt
              A   ENDP

----          EBENE_1_CODE   ENDS

----          EBENE_1_CODE_1 SEGMENT  EO                        ;Nur ausfuehrbares Segment mit
                                                                ;Basisadresse=2000H und DPL=1
                                                                ;(Festlegung durch BLD286)
0000          B   PROC    FAR

0000 55               PUSH    BP                                ;Rette BP der Prozedur A
0001 8BEC             MOV     BP,SP                             ;Stelle BP an die Spitze des
                                                                ;Ebene 1-Stack
0003 C55E06           LDS     BX,[BP].POINTER                   ;(BX)=Pointer-Offset
                                                                ;(DS)=Pointer-Selektor
0006 8A07             MOV     AL,DS:BYTE PTR[BX]                ;(AL)=1.tes Zeichen von ASCII_STRING
0008 5D               POP     BP                                ;Lade BP mit dem alten Prozedur A-Wert
0009 CA0400           RET     4                                 ;Entferne 32-Bit Pointer aus dem
                                                                ;Ebene 1-Stack
              B   ENDP

000C          C   PROC    FAR

000C 55               PUSH    BP                                ;Rette BP der Prozedur A
000D 8BEC             MOV     BP,SP                             ;Stelle BP an die Spitze des
                                                                ;Ebene 1-Stack
000F C55E06           LDS     BX,[BP].POINTER                   ;(BX)=Pointer-Offset
                                                                ;(DS)=Pointer-Selektor
```

```
0012 8807          66        MOV     BYTE PTR[BX],AL          ;(AL) in 1.tes Element von VEKTOR
0014 5D            67        POP     BP                       ;Lade BP mit dem alten Prozedur A-Wert
0015 CA0400        68        RET     4                        ;Entferne 32-Bit Pointer aus dem
                   69                                         ;Ebene 1-Stack
                   70    C   ENDP
                   71
----               72    EBENE_1_CODE_1 ENDS
                   73
----               74    EBENE_2_DATEN  SEGMENT RW            ;Schreib-und lesbares Segment mit
                   75                                         ;Basisadresse=3000H und DPL=2
                   76                                         ;(Festlegung durch BLD286)
0000 (10           77    VEKTOR      DB  10 DUP(?)            ;Vektor mit 10 Byte-Elementen
     ??
     )
                   78
----               79    EBENE_2_DATEN  ENDS
                   80
----               81    EBENE_3_CODE   SEGMENT ER            ;Ausfuehrbares und lesbares Segment
                   82                                         ;mit Basisadresse=4000H und DPL=3
                   83                                         ;(Festlegung durch BLD286)
                   84        ;
                   85        ;                                ;Befehle
                   86        ;
0000 4C6573656E    87    ASCII_STRING  DB 'Lesen'             ;Zu lesender ASCII-String
                   88
----               89    EBENE_3_CODE   ENDS
                   90
                   91        END

ASSEMBLY COMPLETE,   NO WARNINGS,   NO ERRORS
```

3.16.3 Anwendung der Befehle LAR und LSL

- LAR rw,ew ; Load Access Rights Byte

Der im Befehl angegebene zweite Operand ew ist ein Word-Register oder ein Word-Speicheroperand, der den Wert eines Selektors enthält.

Das "Access Rights"-Byte des vom Selektor erreichbaren Deskriptors wird in das ***High***-Byte des ersten Operanden rw geladen, wobei rw ein Word-Register spezifiziert. Das Low-Byte dieses Registers wird dabei auf Null gesetzt.

Das "Zero Flag" wird auf 1 gesetzt, wenn

- der Selektor-Index ***innerhalb*** der Tabellen-Grenze liegt,
- die Privilegebene DPL des ausgewählten Deskriptors ***größer*** oder ***gleich*** der augenblicklichen Privilegebene CPL ist (DPL $\geqslant$ CPL; wie Datenzugriff) und

- die Privilegebene DPL des ausgewählten Deskriptors ***größer*** oder ***gleich*** dem RPL-Wert des Selektors ist (DPL ⩾ Selektor RPL; wie Datenzugriff).

Im anderen Fall wird das "Zero Flag" gelöscht (ZF = 0).

Wenn dieser Befehl im "Real Address Mode" ausgeführt wird, erfolgt eine Programm-Unterbrechung.

- LSL rw,ew ; Load Segment Limit

Der im Befehl angegebene zweite Operand ew ist ein Word-Register oder ein Word-Speicheroperand, der den Wert eines Selektors enthält.

Das "Limit"-Feld (2 Bytes) des vom Selektor erreichbaren Deskriptors wird in den ersten Operanden rw geladen, wobei rw ein Word-Register spezifiziert.

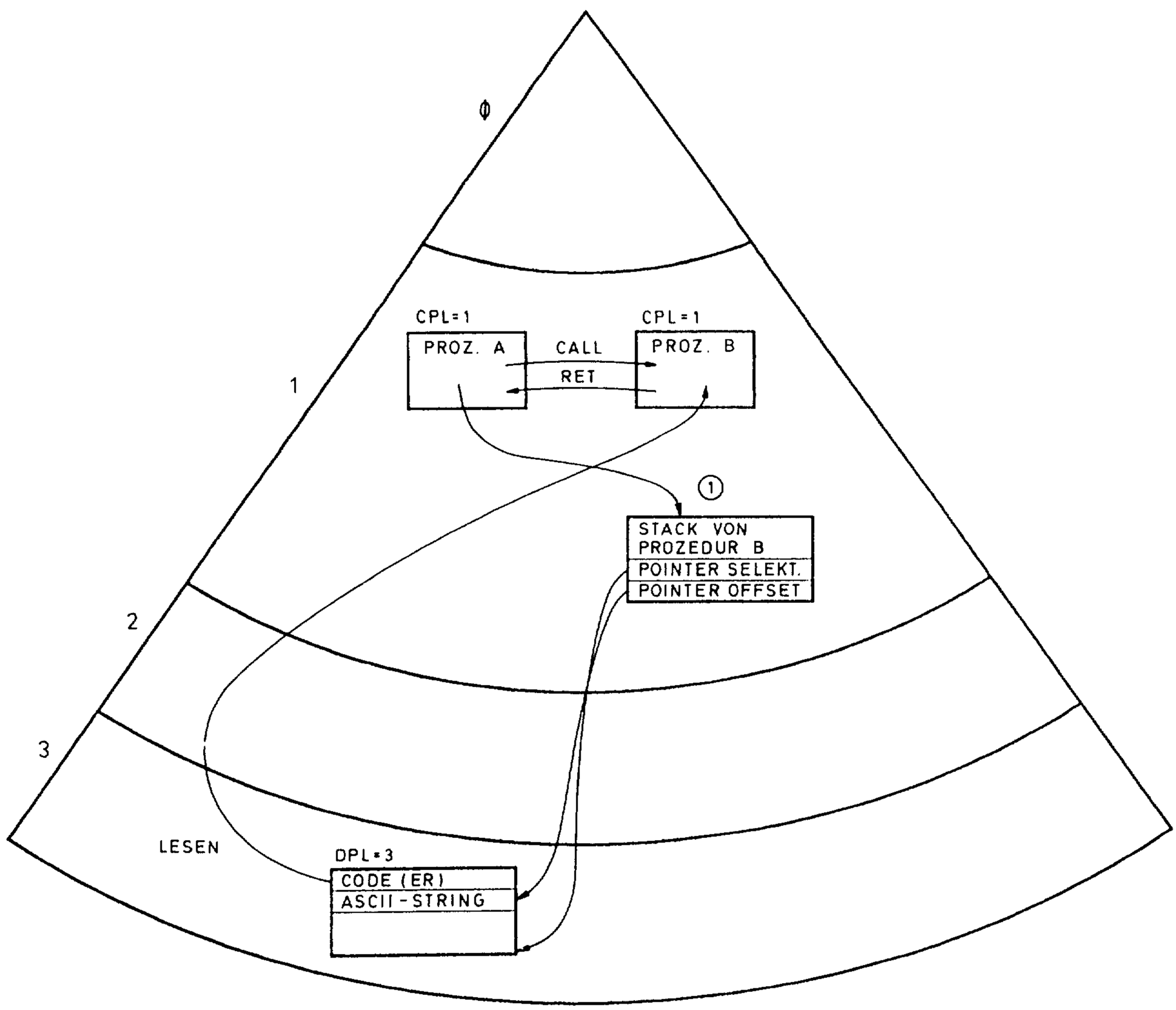
Das "Zero Flag" wird auf 1 gesetzt, wenn

- der Selektor ***nicht*** Null ist,
- der Selektor-Index innerhalb der Tabellen-Grenze liegt, der ausgewählte Deskriptor ein Nicht-"conforming"-Segment-Deskriptor ist, dessen Privilegebene ***größer*** oder ***gleich*** der augenblicklichen Privilegebene CPL ist (DPL ⩾ CPL; wie Datenzugriff) und
- die Privilegebene DPL des ausgewählten Deskriptors ***größer*** oder ***gleich*** dem RPL-Wert des Selektors ist (DPL ⩾ Selektor RPL; wie Datenzugriff).

Im anderen Fall wird das "Zero Flag" gelöscht (ZF = 0).

Wenn dieser Befehl im "Real Address Mode" ausgeführt wird, erfolgt eine Programm-Unterbrechung.

Um zu zeigen, wie die Befehle LAR und LSL benützt werden können, soll eine Situation vorliegen, wie sie das folgende Bild zeigt:



In der Privilegebene 1 existieren die Prozeduren A und B. Dabei benützt A die Prozedur B, um auf ein Nicht-''conforming''-Code-Segment (Typ ER) in der Privilegebene 3 (DPL = 3) zuzugreifen.

Zu diesem Zweck übergibt Prozedur A an den Stack der Prozedur B einen passenden 32-Bit-Pointer ①, den Prozedur B benützt, um einen ASCII-String im Privileg-3-Code-Segment zu lesen.

Das folgende Programmierbeispiel – Zugriffs- und Limitüberprüfung – enthält den ASM286-Code für die angegebene Situation. Dabei wird der Befehl

- LAR DX, AX ; (DX) = "Access"-Byte des
  ; EBENE_3_CODE-Deskriptors

benützt, um später mit

- TEST DH, 00000010B

zu überprüfen, ob das "READABLE"-Bit im "Access"-Byte des EBENE_3_CODE-Segment-Deskriptors gesetzt ist.

Ob das "Access"-Byte überhaupt geladen worden ist, überprüft der Befehl

- JNZ SELEKTOR_UNGUELTIG

Für den Fall, daß der Selektor in AX den Deskriptor ***nicht*** erreicht, weil z. B. die Bedingung DPL ⩾ RPL, CPL ***nicht*** erfüllt ist, wird das "Zero Flag" 0 gesetzt. Der JNZ-Befehl benützt diesen Wert und bewirkt, daß die Prozedur A verlassen wird.

Konnte das "Access"-Byte geladen werden, wertet der Befehl

- JZ NICHT_LESBAR

das Ergebnis des TEST-Befehls aus.
Für den Fall, daß das "READABLE"-Bit 0 ist, kann das EBENE_3_Code-Segment ***nicht*** gelesen werden. Der TEST-Befehl setzt das "Zero Flag" 1. Diesen Wert benützt der JZ-Befehl und bewirkt, daß die Prozedur A verlassen wird.

Ist das EBENE_3_CODE-Segment lesbar, wird anschließend mit

- LSL DI, AX

das "Limit"-Feld des zugehörigen Deskriptors in das DI-Register geladen. Ob das Laden erfolgreich war, ist durch eine Überprüfung des "Zero Flags" an dieser Stelle ***nicht*** mehr notwendig. Daß der Selektor in AX den EBENE_3_CODE-Deskriptor erreichen kann, hat bereits der Befehl JNZ SELEKTOR_UNGUELTIG überprüft.

Der Befehl

- CMP DI, (OFFSET ASCII_STRING + SIZE ASCII_STRING – 1)

stellt nun fest, ob der Wert des "Limit"-Felds mindestens der Größe des EBENE_3_CODE-Segments entspricht. Wenn nein, bewirkt

- JB LIMIT_ZU_KLEIN

daß die Prozedur A verlassen wird.

3.16.4 **Programmierbeispiel:** Zugriffs- und Limitüberprüfung

```
iAPX286 MACRO ASSEMBLER    B15


AiD/UDI V2.2 - iAPX286 MACRO ASSEMBLER V1.1 ASSEMBLY OF MODULE B15
OBJECT MODULE PLACED IN B15.OBJ
ASSEMBLER INVOKED BY:  ASM286.86 B15.SRC


LOC  OBJ                  LINE     SOURCE

                            1            NAME    B15
                            2
----                        3     EBENE_1_CODE    SEGMENT  EO            ;Nur ausfuehrbares Segment mit
                            4                                            ;Basisadresse=1000H und DPL=1
                            5                                            ;(Festlegung durch BLD286)
----                        6     STACK_FRAME     STRUC
0000                        7     PROZEDUR_A_BP       DW  ?             ;Prozedur A-BP
0002                        8     IP1                 DW  ?             ;Ruecksprung-IP
0004                        9     CS1                 DW  ?             ;Ruecksprung-CS
0006                       10     POINTER             DD  ?             ;32-Bit Pointer-Parameter
----                       11     STACK_FRAME     ENDS
                           12
0000                       13     A   PROC    NEAR
                           14
0000 B8----          R     15         MOV     AX,EBENE_3_CODE           ;(AX)=Selektor von EBENE_3_CODE
0003 8ED8                  16         MOV     DS,AX                     ;Selektor laden
0005 0F02D0                17         LAR     DX,AX                     ;(DX)=Access-Byte des EBENE_3_CODE-
                           18                                           ;Deskriptors
0008 7517                  19         JNZ     SELEKTOR_UNGUELTIG        ;Sprung, wenn ungueltiger Selektor
                           20                                           ;(Zero Flag=0)
000A F6C602                21         TEST    DH,00000010B              ;Ist READABLE-Bit gesetzt?
000D 7412                  22         JZ      NICHT_LESBAR              ;Sprung, wenn READABLE-Bit=0
                           23                                           ;(Zero Flag=1)
000F 0F03F8                24         LSL     DI,AX                     ;(DI)=Limit des EBENE_3_CODE-Segments
0012 83FF0690              25         CMP     DI,(OFFSET ASCII_STRING+SIZE ASCII_STRING-1)
                           26                                           ;Limit-Wert mit Segment-Groesse
                           27                                           ;vergleichen
0016 7209                  28         JB      LIMIT_ZU_KLEIN            ;Sprung, wenn der Limit-Wert zu klein
                           29                                           ;ist
0018 1E                    30         PUSH    DS                        ;Pointer-Selektor in Ebene 1-Stack
0019 6A0290                31         PUSH    OFFSET ASCII_STRING       ;Pointer-Offset in Ebene 1-Stack
001C 9A0000----      R     32         CALL    FAR PTR B                 ;Aufruf von Prozedur B inter-segment
                           33                                           ;direkt
                           34         ;
                           35         ;
0021                       36     SELEKTOR_UNGUELTIG:
0021                       37     NICHT_LESBAR:
0021                       38     LIMIT_ZU_KLEIN:
0021 C3                    39         RET                               ;Prozedur A verlassen
                           40
```

```
                        41      A   ENDP
                        42
----                    43      EBENE_1_CODE   ENDS
                        44
----                    45      EBENE_1_CODE_1 SEGMENT  EO              ;Nur ausfuehrbares Segment mit
                        46                                              ;Basisadresse=2000H und DPL=1
                        47                                              ;(Festlegung durch BLD286)
0000                    48      B   PROC    FAR
                        49
0000 55                 50          PUSH    BP                          ;Rette BP der Prozedur A
0001 8BEC               51          MOV     BP,SP                       ;Stelle BP an die Spitze des
                        52                                              ;Ebene 1-Stack
0003 C55E06             53          LDS     BX,[BP].POINTER             ;(BX)=Pointer-Offset
                        54                                              ;(DS)=Pointer-Selektor
0006 8A07               55          MOV     AL,DS:BYTE PTR[BX]          ;(AL)=1.tes Zeichen von ASCII_STRING
0008 5D                 56          POP     BP                          ;Lade BP mit dem alten Prozedur A-Wert
0009 CA0400             57          RET     4                           ;Entferne 32-Bit Pointer aus dem
                        58                                              ;Ebene 1-Stack
                        59      B   ENDP
                        60
----                    61      EBENE_1_CODE_1 ENDS
                        62
----                    63      EBENE_3_CODE   SEGMENT ER               ;Ausfuehrbares und lesbares Segment
                        64                                              ;mit Basisadresse=4000H und DPL=3
                        65                                              ;(Festlegung durch BLD286)
                        66          ;
                        67          ;                                   ;Befehle
                        68          ;
0000 90                 69          NOP
0001 90                 70          NOP
0002 4C6573656E         71      ASCII_STRING  DB 'Lesen'                ;Zu lesender ASCII-String
                        72
----                    73      EBENE_3_CODE   ENDS
                        74
                        75          END

ASSEMBLY COMPLETE,   NO WARNINGS,   NO ERRORS
```