7 Eingabe/Ausgabe

7.1 Bus-Transfer-Mechanismus

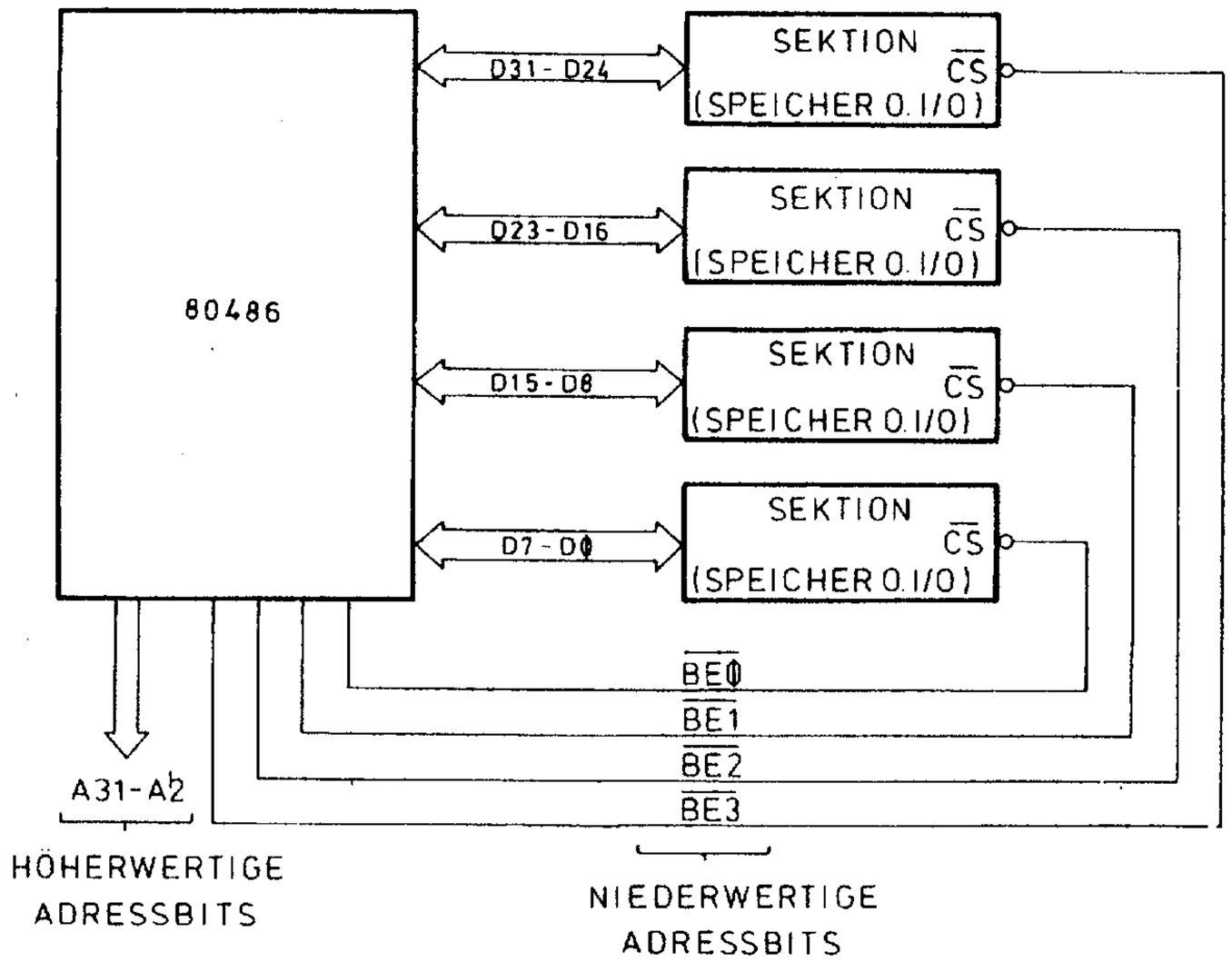
Der Programmierer sieht den Adreßraum (Speicher und I/O) des 80486 als eine Sequenz von Bytes. Dabei bestehen Words aus ***zwei benachbarten*** Bytes und Doublewords aus ***vier benachbarten*** Bytes. Im Gegensatz dazu ist in der System-Hardware der Adreßraum in 4 Speicher- und I/O-Sektionen unterteilt. Hier ist jeder der vier 8-Bit-Teile des Datenbus (D0 ... D7, D8 ... D15, D16 ... D23 und D24 ... D31) mit einer individuellen Sektion verbunden.

Immer dann, wenn der 80486 ein Doubleword liest, greift er in jeder Sektion auf ein einzelnes Byte zu. Damit diese Hardware-Implementation einfach gehalten werden kann, unterstützt sie der 80486 durch die höherwertigen Adreß-Bits ***A2 ... A31*** und die niederwertigen Adreß-Bits $\overline{\textbf{\textit{BE0}}}$ ... $\overline{\textbf{\textit{BE3}}}$. Dabei sind den „Byte Enable"-Signalen $\overline{\textbf{BE0}}$ ... $\overline{\textbf{BE3}}$ bestimmte Datenbus-Leitungen zugeordnet.

Diese Zuordnungen sind in der folgenden Tabelle zu sehen. Sie bedeuten, daß der Prozessor abhängig von den beanspruchten Datenbus-Bytes, die entsprechenden „Byte Enable"-Ausgänge aktiviert (Low-Signal).

Byte Enable-Signale	Zugeordnete Datenbus-Leitungen
$\overline{\textbf{BE0}}$	D0 ... D7 (Byte 0)
$\overline{\textbf{BE1}}$	D8 ... D15 (Byte 1)
$\overline{\textbf{BE2}}$	D16 ... D23 (Byte 2)
$\overline{\textbf{BE3}}$	D24 ... D31 (Byte 3)

Die „***B***yte ***E***nable"-Signale $\overline{\textbf{BE0}}$ ... $\overline{\textbf{BE3}}$ liefern direkt die linearen „Chip Selects" für die ***vier*** Bytes des 32-Bit-Datenbus. Wie diese Signale benützt werden können, um in der System-Hardware benachbarte Bytes (so wie sie der Programmierer sieht) auszuwählen, ist im folgenden Bild illustriert:



Damit der Prozessor die passenden „Byte Enable"-Ausgänge aktivieren kann, benutzt er ***intern*** die Adreß-Signale A0 und A1. Sie stehen daher extern ***nicht*** zur Verfügung. Wie die in den physikalischen Basis-Adressen enthaltenen A1/A0-Signalkombinationen mit den „Byte Enable"-Ausgängen korrespondieren, zeigt die folgende Tabelle:

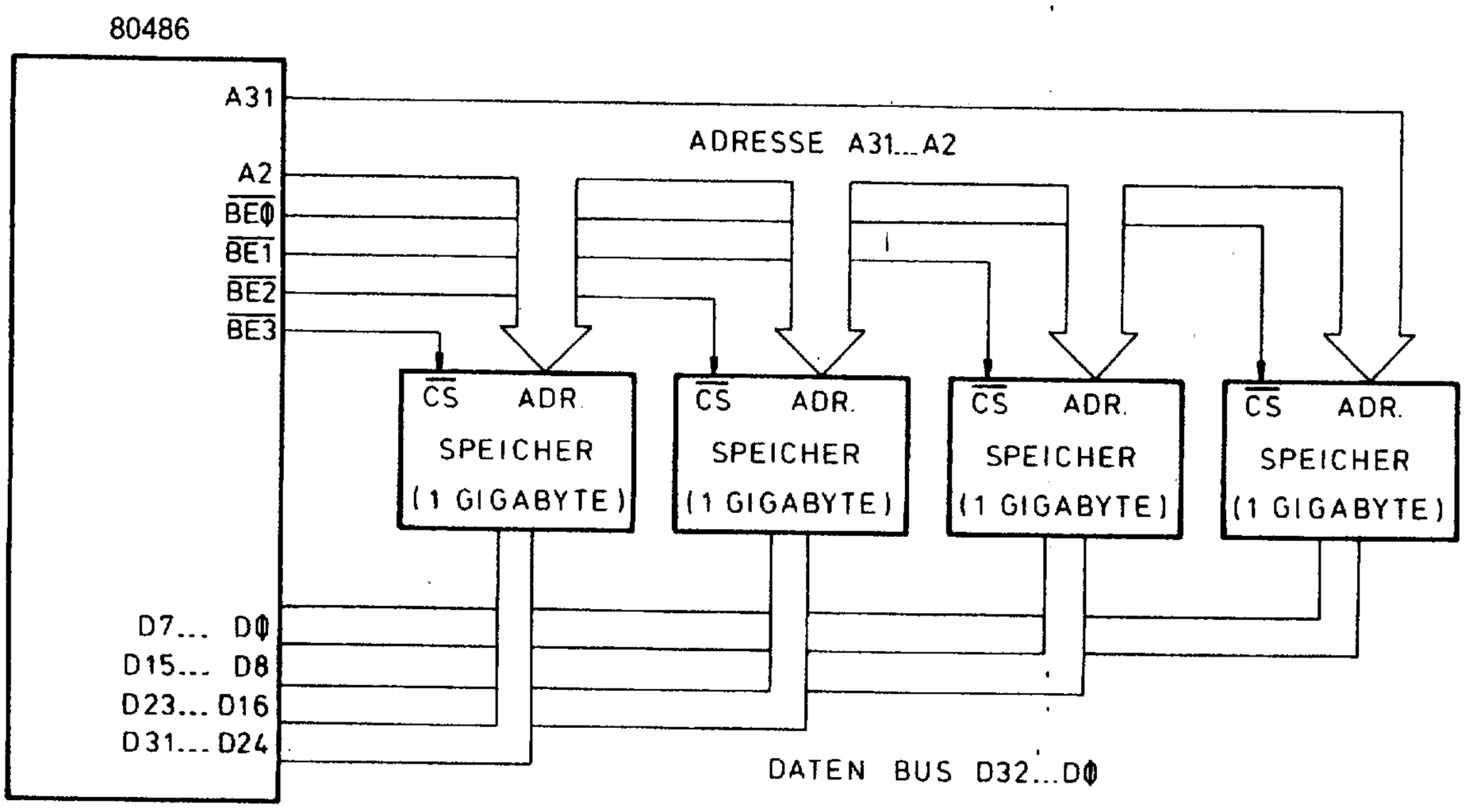
80486-Adreß-Signale								
A31	.....	A2			$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$
	Physikalische Basis-Adressen							
A31	.....	A2	A1	A0				
A31	.....	A2	0	0	x	x	x	Low
A31	.....	A2	0	1	x	x	Low	High
A31	.....	A2	1	0	x	Low	High	High
A31	.....	A2	1	1	Low	High	High	High

Prozessor – intern benutzt

Dabei kennzeichnen die A1/A0-Werte die Position der individuellen Bytes in einem Doubleword. Immer dann, wenn der Prozessor eine bestimmte physikalische Basis-Adresse liefert, aktiviert er automatisch den zum A1/A0-Wert zugehörigen „Byte Enable"-Ausgang. Dadurch wird in der System-Hardware die der Byte-Position zugeordnete Sektion immer richtig ausgewählt.

7.1.1 Speicher-Selektion

Der 80486 kann bis zu 4 Gigabytes (Adressen 00000000H ... FFFFFFFFH) im physikalischen Speicher adressieren. Wie zum Beispiel ein Speicherraum von 4 Gigabytes prinzipiell konfiguriert werden kann, zeigt das folgende Bild:

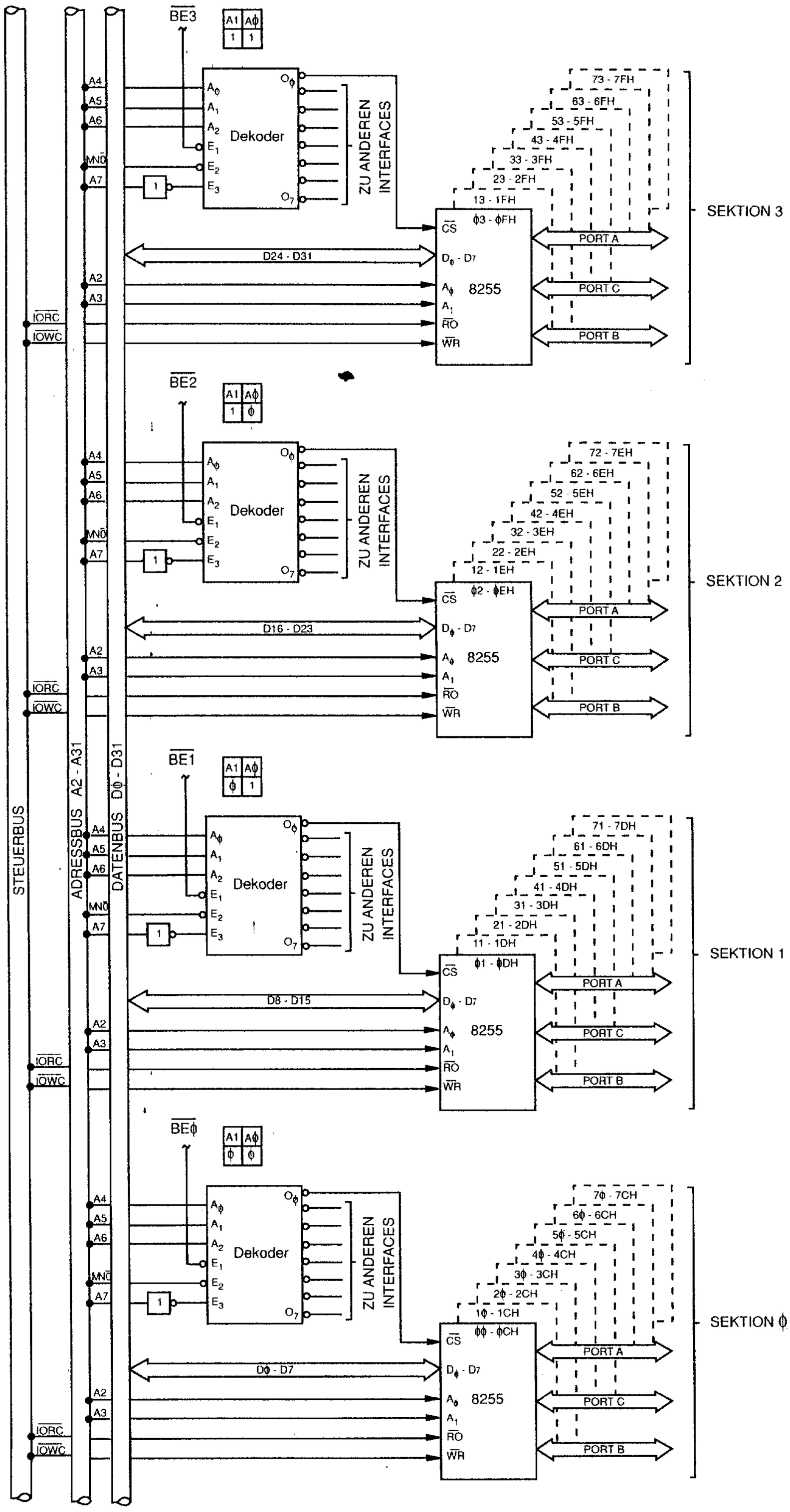


Es existieren 4 Speicher-Sektionen zu je 1 Gigabyte, die jeweils mit einem benachbarten 8-Bit-Teil des 32-Bit-Datenbus verbunden sind. Dadurch ist jede Sektion durch eine spezifische A1/A0-Nummer gekennzeichnet, wobei diese Nummer mit der Position eines Bytes innerhalb eines Doublewords identisch ist.

Wenn nun der Prozessor eine bestimmte physikalische Byte-Adresse liefert, aktiviert er den zum A1/A0-Wert gehörigen „Byte Enable"-Ausgang und selektiert damit die richtige Speicher-Sektion. Die Byte-Position innerhalb des selektierten Speichers wird dann durch die 30 externen Adreß-Signale A31 ... A2 bestimmt, die maximal einen Raum von 1 Gigabyte adressieren können.

7.1.2 I/O-Selektion

Der 80486 kann bis zu 64 KBytes (Adressen 0000H ... FFFFH) im I/O-Raum adressieren. Wie zum Beispiel ein I/O-Raum für 32-Byte-Interfaces konfiguriert werden kann, zeigt das folgende Bild:


BE3
A1 Aϕ
1 1
A4
A5
A6
MN0
A7
Aϕ
A1
A2
E1
E2
E3
Dekoder
Oϕ
O7
1
ZU ANDEREN INTERFACES
73 - 7FH
63 - 6FH
53 - 5FH
43 - 4FH
33 - 3FH
23 - 2FH
13 - 1FH
ϕ3 - ϕFH
SEKTION 3
CS
Dϕ - D7
8255
RO
WR
PORT A
PORT C
PORT B
D24 - D31
A2
A3
IORC
IOWC
BE2
A1 Aϕ
1 ϕ
72 - 7EH
62 - 6EH
52 - 5EH
42 - 4EH
32 - 3EH
22 - 2EH
12 - 1EH
ϕ2 - ϕEH
SEKTION 2
D16 - D23
BE1
A1 Aϕ
ϕ 1
71 - 7DH
61 - 6DH
51 - 5DH
41 - 4DH
31 - 3DH
21 - 2DH
11 - 1DH
ϕ1 - ϕDH
SEKTION 1
D8 - D15
BEϕ
A1 Aϕ
ϕ ϕ
7ϕ - 7CH
6ϕ - 6CH
5ϕ - 5CH
4ϕ - 4CH
3ϕ - 3CH
2ϕ - 2CH
1ϕ - 1CH
ϕϕ - ϕCH
SEKTION ϕ
Dϕ - D7
STEUERBUS
ADRESSBUS A2 - A31
DATENBUS Dϕ - D31

Es existieren 4 I/O-Selektionen mit je 8 parallelen Ein-/Ausgabe-Bausteinen (8255), die jeweils mit einem benachbarten 8-Bit-Teil des 32-Bit-Datenbus verbunden sind. Dadurch ist jede Sektion durch eine spezifische A1/A0-Nummer gekennzeichnet. Wenn nun der Prozessor eine bestimmte Port-Adresse liefert, aktiviert er den zum A1/A0-Wert gehörigen „Byte Enable"-Ausgang.

In der Konfiguration ist zu erkennen, daß jeder I/O-Sektion ein einzelner Binär-Dekoder als sogenannter „Chip Select"-Generator zugeordnet ist. Immer dann, wenn für die individuellen „Chip Select"-Generatoren die erforderlichen Eingabe-Bedingungen erfüllt sind, können die ihnen zugeordneten I/O-Sektionen adressiert werden. Diese Bedingungen sind in der folgenden Liste zusammengefaßt:

$M/\overline{IO}$	$\overline{A7}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	Freigabe des Chip Select-Generators für
Low	Low	x	x	x	Low	Sektion 0
Low	Low	x	x	Low	High	Sektion 1
Low	Low	x	Low	High	High	Sektion 2
Low	Low	Low	High	High	High	Sektion 3

- $\overline{BE0}$ → A1/A0 = 00B
- $\overline{BE1}$ → A1/A0 = 01B
- $\overline{BE2}$ → A1/A0 = 10B
- $\overline{BE3}$ → A1/A0 = 11B
- A7 → Das Systembus-Adreß-Signal A7 und die prozeßorientierten Signale A1/A0 bestimmen zusammen die Basis-Adressen der individuellen I/O-Sektionen.
- $M/\overline{IO}$ → Freigabe der „Chip Select"-Generatoren nur durch I/O-Befehle.

Da jede I/O-Sektion aus 8 einzelnen Bausteinen besteht, werden in der angegebenen Konfiguration die Systembus-Adreß-Signale A4, A5 und A6 benutzt, um die erforderliche „Chip Select"-Leitung 0_0 ... 0_7 für den gewünschten Baustein in der ausgewählten Sektion zu aktivieren.

Um nun im selektierten Baustein die einzelnen Ein-/Ausgabe-Ports (A, B, C oder Steuerport) zu erreichen, liefern die System-Adreß-Leitungen A3/A2 die passenden Signal-Kombinationen.

Die folgende Wahrheitstabelle soll die angegebene Dekodier-Methode verdeutlichen:

80486-Adreß-Signale													
A15 . . . . . A2									M/$\overline{IO}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$
Physikalische Basis-Adressen										Selektion 3	Selektion 2	Selektion 1	Selektion 0
A15 ...	A7	A6	A5	A4	A3	A2	A1	A0					
A15 ...	0	x	x	x	x	x	0	0	Low	x	x	x	Low
A15 ...	0	x	x	x	x	x	0	1	Low	x	x	Low	High
A15 ...	0	x	x	x	x	x	1	0	Low	x	Low	High	High
A15 ...	0	x	x	x	x	x	1	1	Low	Low	High	High	High

- Aktiv bei I/O-Befehlen
- Aktiviert $\overline{BE0}$ bis $\overline{BE3}$
- Adressierung von Port A, B, C oder Steuerport
- Aktivierung der „Chip-Select"-Leitungen $0_0 - 0_7$
- Mit A1/A0 die Basis-Adressen der ausgewählten Sektion festlegen
- A8 bis A15 ist nicht dekodiert. Soll 00H sein.

BEISPIEL:

In der Sektion 0 soll der Port A im letzten Baustein (Nummer 7) adressiert werden. In diesem Fall liefert der Prozessor die folgende Adreß-Signale:

M/$\overline{IO}$	$\overline{BE0}$	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
L	L	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	B = 0070H

- M/$\overline{IO}$ = L: Freigabe der Sektion 0 durch I/O-Befehl
- $\overline{BE0}$ = L: Sektion 0 freigeben
- A15 ... A8 = 0: Nicht dekodiert, Soll 00H sein
- A7 = 0: Bestimmt mit A1/A0 die Basisadresse der Sektion 0
- A6 ... A4 = 1: Baustein 7 selektieren
- A3 ... A2 = 0: Port A adressieren
- A1 ... A0 = 0: BE0 aktivieren

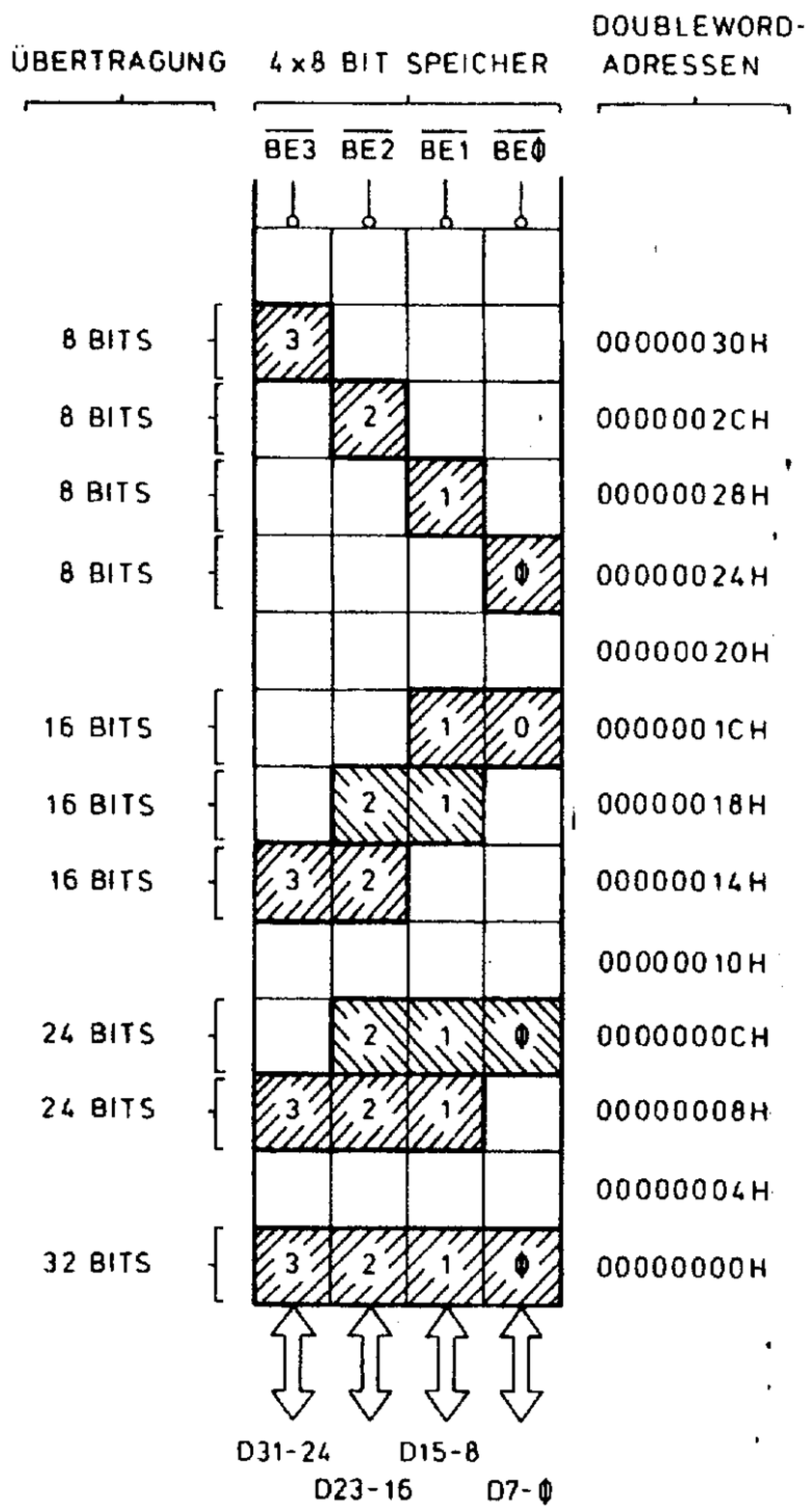
Mit der Befehls-Sequenz

- **MOV DX,0070H** ; DX = Adresse von Port A im letzten Baustein
  ; der Sektion 0
- **IN AL,DX** ; Port A lesen

kann nun Port A zum Beispiel gelesen werden.

7.1.3 32-Bit-Datenbus-Transfer und Ausrichtung der Operanden

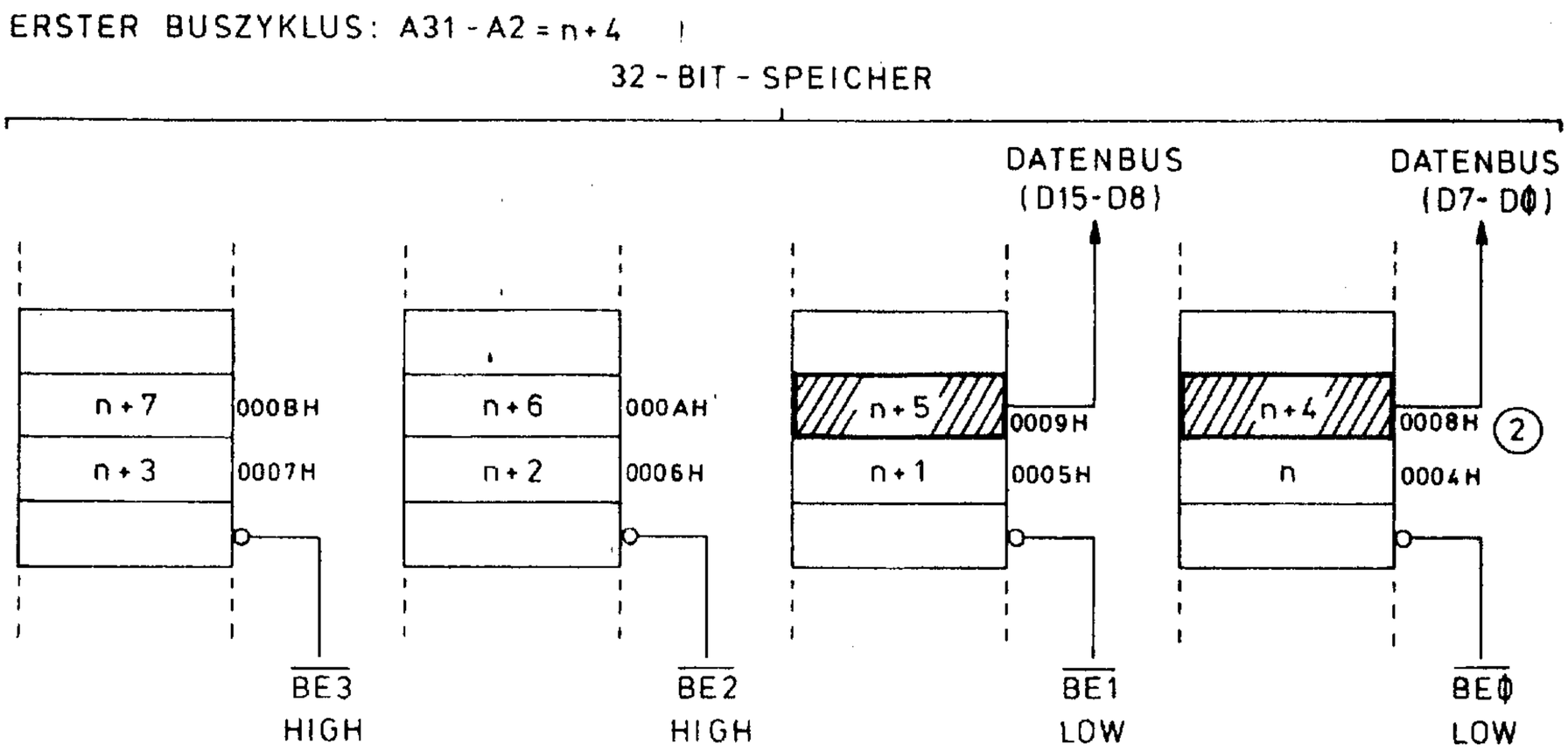
Der 80486 kann Daten in Einheiten von 32 Bits, 24 Bits, 16 Bits oder 8 Bits ***in einem einzigen Bus-Zyklus*** transferieren. Wenn ein Daten-Transfer in einem einzelnen Zyklus erledigt werden kann, wird er als „ausgerichtet" (aligned) bezeichnet. Solche Transfers sind im folgenden Bild illustriert. Dabei kennzeichnen die Nummern 0, 1, 2 und 3 die Byte-Positionen innerhalb der Übertragungs-Einheiten.

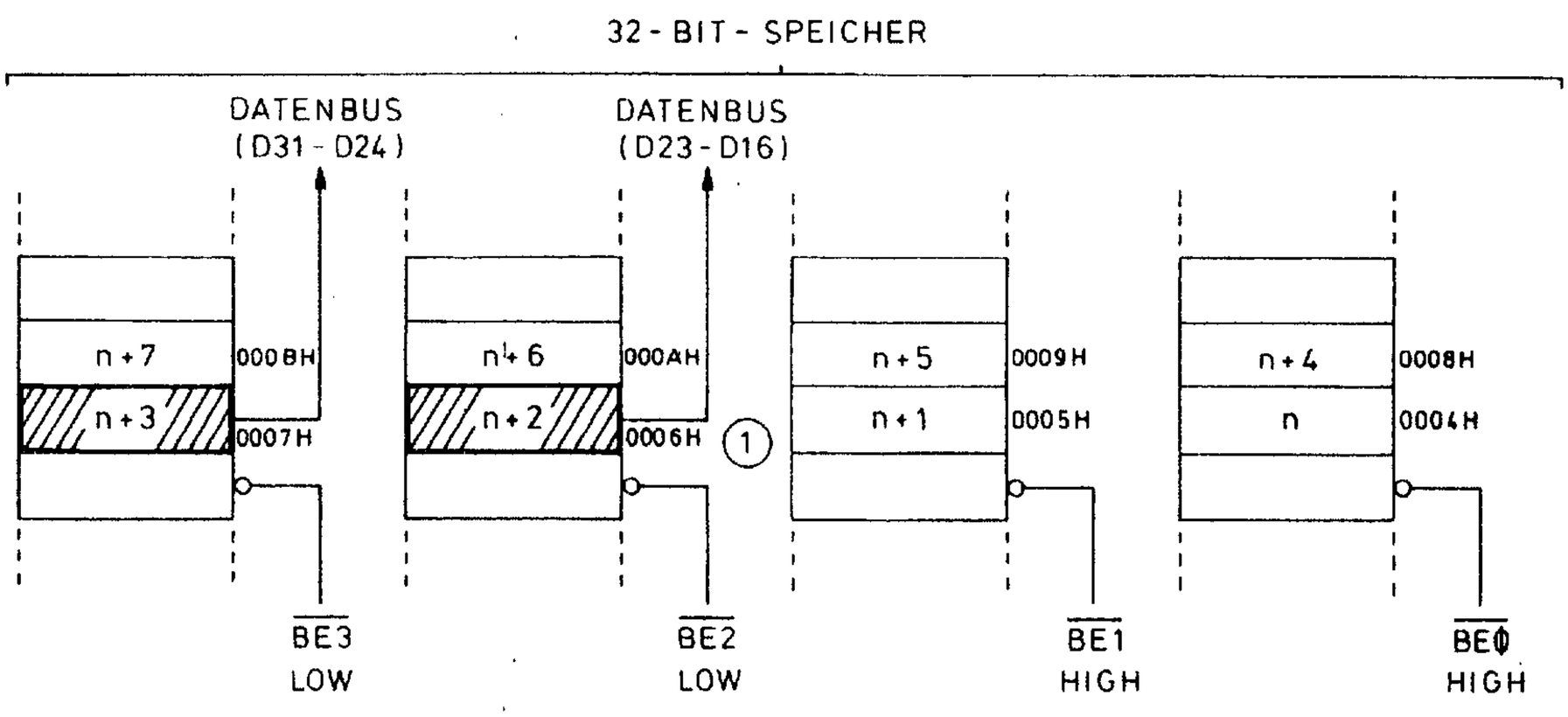


Wenn ein „Word"- oder „Doubleword"-Transfer die „Doubleword"-Grenze (A1/A0 = 00B) des 80486 überschreitet, wird er als „nicht-ausgerichtet" (misaligned) bezeichnet. Für einen solchen Transfer sind immer 2 Bus-Zyklen erforderlich, die der Prozessor automatisch generiert.

Die folgenden Bilder zeigen als Beispiel die für einen „nicht-ausgerichteten" 32 Bit-Transfer erforderliche Schritte. Dabei soll die „Doubleword"-Grenze n (A31 ... A2) = 0004H sein (A1/A0 = 00B) und der 32 Bit-Transfer ab der Adresse n+2 (A31 ... A2) = 0006H (A1/A0 = 10B) beginnen ①.

Im ***ersten*** Bus-Zyklus liefert der Prozessor die Adresse n+4 (A31 ... A2) = 0008H (A1/A0 = 00B) und kennzeichnet damit die nächste „Doubleword"-Position ②. Da bei diesem Zyklus $\overline{BE0}$ und $\overline{BE1}$ aktiv sind (Low), erfolgt ein „Word"-Transfer über die Datenleitungen D0 ... D7 und D8 ... D15.





Im ***zweiten*** Bus-Zyklus dekrementiert der Prozessor die Adresse und liefert den Wert n+2 (A31 ... A2) = 0006H (A1/A0 = 10B). Bei diesem Zyklus sind $\overline{BE2}$ und $\overline{BE3}$ aktiv (Low), so daß ein „Word"-Transfer über die Datenleitungen D23 ... D16 und D31 ... D24 erfolgt.

Wie das Beispiel zeigt, wird bei einem „grenzüberschreitenden" „Doubleword"-Transfer zuerst das „High Word" (Adresse 00000008H) über den niederwertigen Teil des 32 Bit-Datenbus (D15 ... D0) und anschließend das „Low Word" (Adresse 00000006H) über den höherwertigen Teil des 32 Bit-Datenbus (D31 ... D16) übertragen. Dieser Widerspruch wird allerdings nach dem Transfer vom Prozessor ***intern*** bereinigt. Dies bedeutet, daß die

Daten-Bits in die richtige Ordnung gebracht werden, so daß zum Beispiel nach einem Befehl wie ***MOV reg32, mem32*** erwartungsgemäß das „Low Word" im niederwertigen Teil des Registers und das „High Word" im höherwertigen Teil des Registers zu finden sind.

Der angegebene Mechanismus gilt auch für „nicht-ausgerichtete" „Word"-Transfer. Erfolgt zum Beispiel ein Word-Zugriff auf die Byte-Adresse 0003H, sind ***zwei*** Byte-Transfers erforderlich. Im ersten Bus-Zyklus aktiviert der Prozessor die „Doubleword"-Adresse 0004H und benützt D7 ... D0. Im zweiten Bus-Zyklus aktiviert er die „Doubleword"-Adresse 0000H und benützt D31 ... D24.

Da der 80486 nur mit Bytes, Words und Doublewords operiert, werden bestimmte Kombinationen von $\overline{BE3}$ ... $\overline{BE0}$ niemals produziert. Zum Beispiel wird ***kein*** Bus-Zyklus generiert, wenn $\overline{BE0}$ und $\overline{BE2}$ aktiv sind. Das ist deswegen so, weil ein solcher Transfer eine Operation mit zwei ***nicht***-benachbarten Bytes zur gleichen Zeit wäre.

Die folgende Tabelle enthält alle $\overline{BE3}$ ... $\overline{BE0}$-Signalkombinationen und zeigt auch, welche Bit-Muster ***nicht*** vorkommen können.

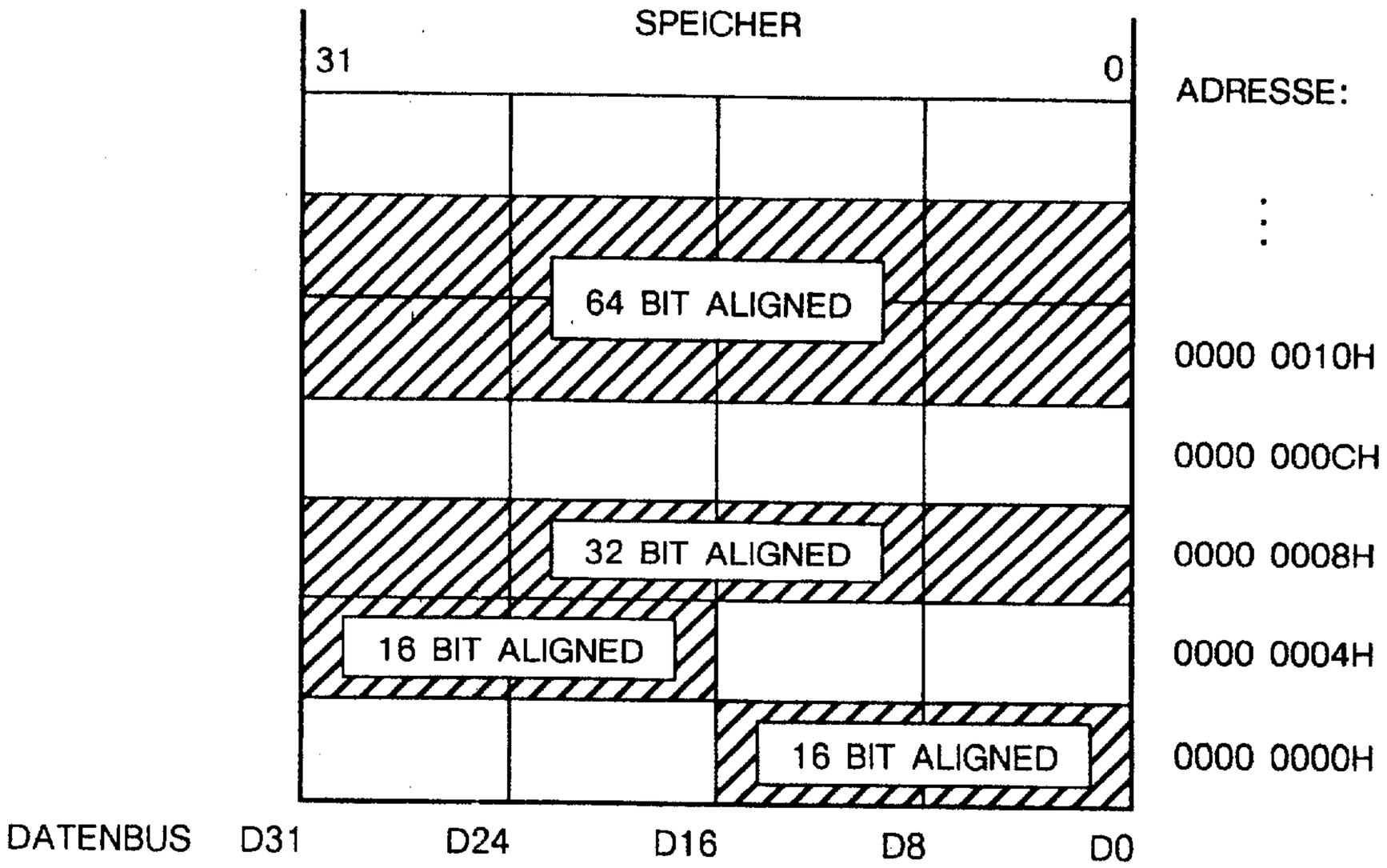
80486-Signale				Kommentar
$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	
H*	H*	H*	H*	* – keine aktiven Bytes
H H H H H	H H H H L	H L L L H	L H L L H	
H*	L*	H*	L*	* – keine benachbarten Bytes
H H L	L L H	L L H	H L H	
L*	H*	H*	L*	* – keine benachbarten Bytes
L*	H*	L*	H*	* – keine benachbarten Bytes
L*	H*	L*	L*	* – keine benachbarten Bytes
L	L	H	H	
L*	L*	H*	L*	* – keine benachbarten Bytes
L L	L L	L L	H L	

7.1.4 Alignment-Überprüfung

Wenn 80486-Operanden mit einem Minimum an Transferzyklen gelesen oder geschrieben werden sollen, dann müssen sie an ihren natürlichen Adreßgrenzen stehen und zwar

- 16 Bit-Operanden an Adressen mit A0 = 0,
- 32-Bit-Operanden an Adressen mit A1/A0 = 00 und
- 64-BIT-Operanden an Adressen mit A2/A1/A0 = 000.

Im folgenden Speicher-Layout sind einige natürliche Positionen dieser Operanden illustriert:

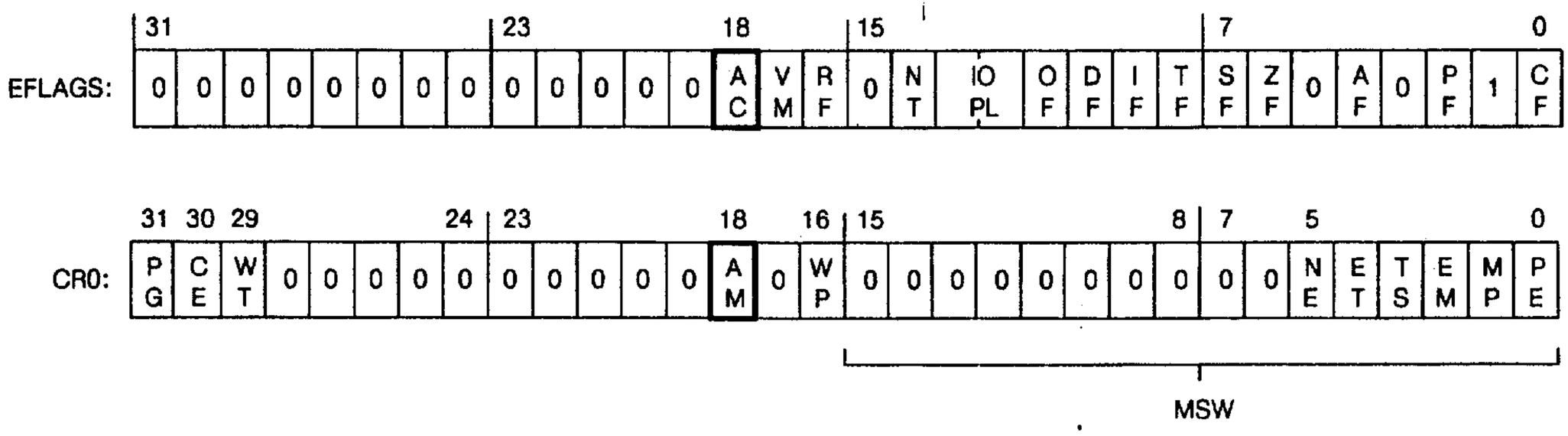


Der 80486 unterstützt nun eine ganze Reihe von Datentypen, die alle durch individuelle Adreßgrenzen gekennzeichnet sind. Um welche Datentypen es sich dabei handelt und wie sie im Speicher ausgerichtet sein sollten, zeigt die folgende Tabelle:

Erforderliche Datentyp-Ausrichtung im Speicher

Datentyp	Speicheradresse teilbar durch
Word	2
Doubleword	4
Short-Real	4
Long-Real	8
Temporary-Real	8
Selektor	2
48-Bit Segment-Pointer	4
32-Bit Segment-Pointer (im flachen Adreßraum)	4
32-Bit Segment-Pointer	2
48-Bit Pseudo-Deskriptor	4
FSTENV/FLDENV (Speichere/Lade FPU-Environment)	4 (USE32-Segment) 2 (USE16-Segment)
FSAVE/FRSTOR (Speichere/Lade FPU-Status)	4 (USE32-Segment) 2 (USE16-Segment)
Bit-String	4

Wenn der Prozessor einen Operanden transferiert, der **nicht** an seiner natürlichen Adreßgrenze steht, dann benötigt er für einen solchen Transfer mehr als nur einen Zyklus. Beispiele hierfür sind Word-Zugriffe auf **ungerade** Adressen oder Doubleword-Zugriffe auf Adressen, die **nicht** durch **4** teilbar sind. Die Folge ist, daß sich die Leistung des Prozessors um so mehr verringert, je häufiger solche Zugriffe vorkommen. Aus diesem Grund hat der 80486 einen Mechanismus, mit dessen Hilfe alle diejenigen Programmstellen "aufgespürt" werden können, bei denen Zugriffe auf nicht-ausgerichtete Operanden erfolgen. Aktiviert wird dieser Mechanismus vom **AC**-Bit (**A**lignment **C**heck) in den **EFLAG**s und vom **AM**-Bit (**A**lignment **M**ask) im **CR0**-Register. Die Positionen dieser Bits und ihr Zusammenwirken sind den folgenden Layouts und in der Tabelle zu sehen:



AM	AC	Alignment-Überprüfung
0	0	Nein
0	1	Nein
1	0	Nein
1	1	Ja

Kommt es während des Programmablaufs zu einem Zugriff auf einen **nicht**-ausgerichteten Operanden und sind dabei **AM/AC = 11** (binär), dann wird **Interrupt 17** ausgelöst. Der aufgerufene Interrupt-Handler kann daraufhin die Fehlplatzierung beheben, so daß bei künftigen Zugriffen auf diesen Operanden der Interrupt nicht mehr generiert wird. Weil bei dieser Ausnahme die CS- und EIP-Werte des fehlerverursachenden Befehls in den Stack gerettet werden, ist eine anschließende Wiederholung des Befehls möglich.

Die Aligment-Überprüfung wirkt nur in Programmen, die in der Privileg-Ebene **3** ablaufen. Das AC-Bit ist **unwirksam** in den Privileg-Ebenen 0,1,2 und bei Referenzen auf Deskriptor-Tabellen oder Task-Status-Segmente, selbst dann, wenn diese Referenzen in der Privileg-Ebene 3 erfolgen.

7.2 Ein-/Ausgabe und Schutz

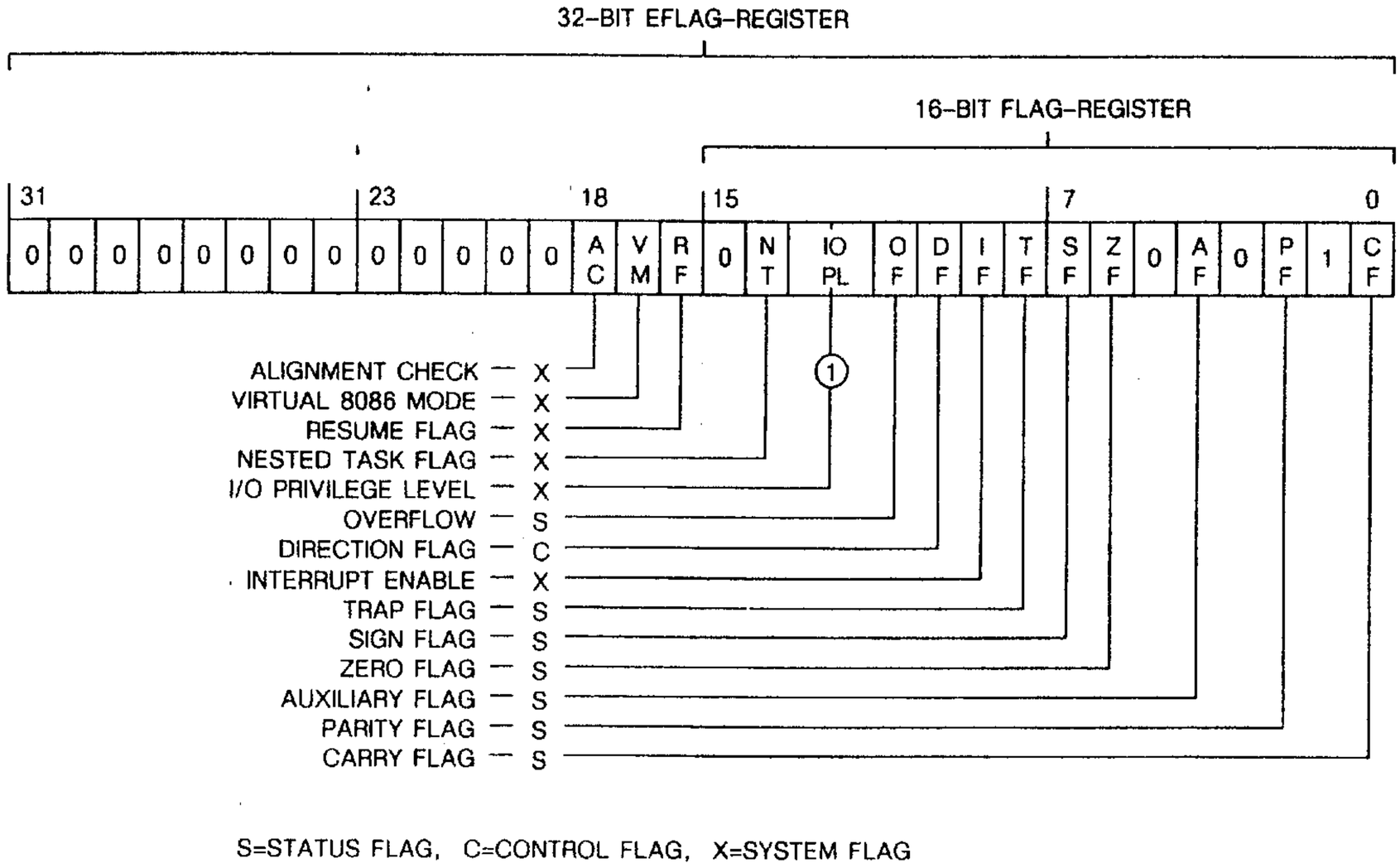
Viele Konzepte, die in den vorherigen Kapiteln eingeführt worden sind, können benützt werden, um im Betriebssystem ein Ein-/Ausgabe-Subsystem zu implementieren.
Dabei sind die Ein-/Ausgabe-Operationen folgendermaßen klassifiziert:

- Die ***direkte*** Ein-/Ausgabe, bei der die 80486-CPU direkt mit den Peripherie-Komponenten kommuniziert. Sie kann unterteilt werden in das
  - „Memory Mapped"-Zugriffsverfahren
  - „I/O Mapped"-Zugriffsverfahren.

  Bei „Memory Mapped" werden Ein-/Ausgabe-Einheiten von Prozessor-Befehlen getriggert, die sich normalerweise auf bestimmte ***Speicher***-Positionen beziehen.
  Bei „I/O Mapped" benützt der Prozessor für die Ein-/Ausgabe-Operationen spezielle Befehle wie IN, OUT, INS und OUTS.
  Jede Position im I/O-Raum wird als I/O-Port bezeichnet. Dabei können Ports als 8-, 16- oder 32-Bit-Einheiten konfiguriert sein. Die Befehle IN und OUT transferieren Bytes, Word oder Doublewords zwischen den Registern AL/AX/EAX und den I/O-Ports. Die Befehle INS und OUTS transferieren Byte-, Word- oder Doubleword-Strings zwischen einem I/O-Port und dem Speicher.
- Die ***indirekte*** Ein-/Ausgabe, bei der ein externer Prozessor (wie z. B. der 82258 Advanced DMA Controller) die Ein-/Ausgabe-Operationen übernimmt.

Das Konzept des Ein-/Ausgabe-Schutzes steht nicht nur in Verbindung mit dem bei Ein/Ausgabe-Operationen benützten Speicher, sondern auch mit dem Recht, ob Ein-/Ausgabe-Operationen durchgeführt werden dürfen.

So ist es möglich, Ein-/Ausgabe-Operationen und ein-/ausgabebezogene Befehle auf bestimmte Privileg-Ebenen zu begrenzen. Zu diesem Zweck steht das IOPL-Feld (I/O-Privilege Level) in den EFLAGS der 80486-CPU zur Verfügung ①.



Dieses Feld beschränkt das Recht einer „Task“, einen der folgenden Befehle auszuführen:

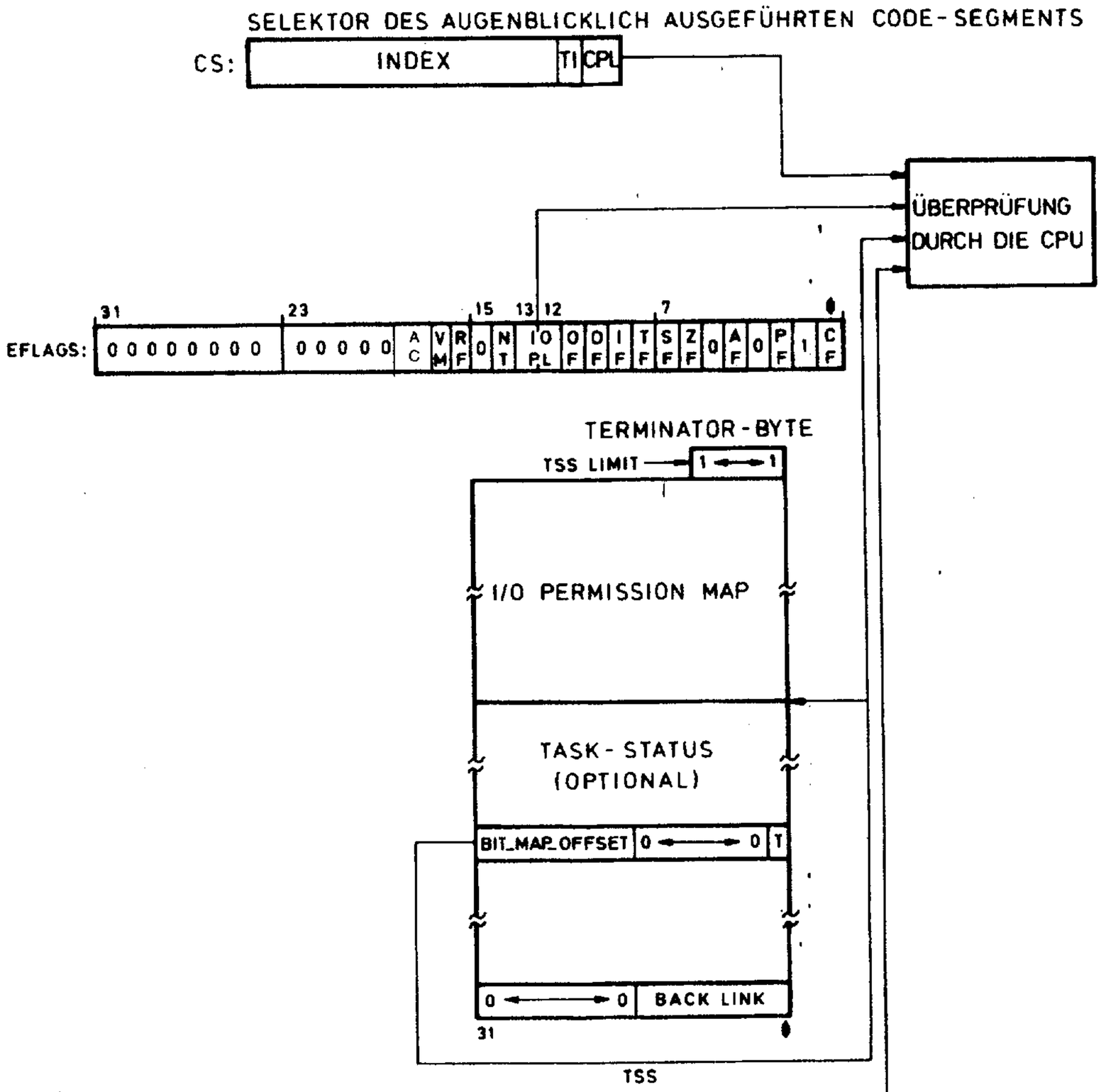
- IN ; Eingabe
- INS ; String-Eingabe
- OUT ; Ausgabe
- OUTS ; String-Ausgabe
- STI ; Setze Interrupt-Flag (Interrupt-Freigabe)
- CLI ; Lösche Interrupt-Flag (Interrupt-Sperre)

7.2.1 I/O-Privileg und „I/O-Permission Bit Map“

Wenn die CPU einen dieser sogenannten „I/O-sensitiven“ Befehle interpretiert, vergleicht sie

1. die augenblickliche Privileg-Ebene CPL der „Task“ mit IOPL in den EFLAGS und
2. das Segment-Limit-Feld im TSS-Deskriptor mit dem BIT_MAP_OFFSET-Feld im Task-Status-Segment.

Dies zeigt das folgende Bild:



Wenn die CPU auf einen I/O-Befehl trifft, prüft sie zunächst, ob die Bedingung

- „Task“ CPL ≤ Peripherie IOPL

erfüllt ist.

Sie bedeutet, daß ein „I/O-sensitiver“ Befehl ***uneingeschränkt*** ausgeführt werden kann, wenn er sich auf einer Privileg-Ebene aufhält, die numerisch ***kleiner*** oder ***gleich*** dem IOPL-Wert ist.

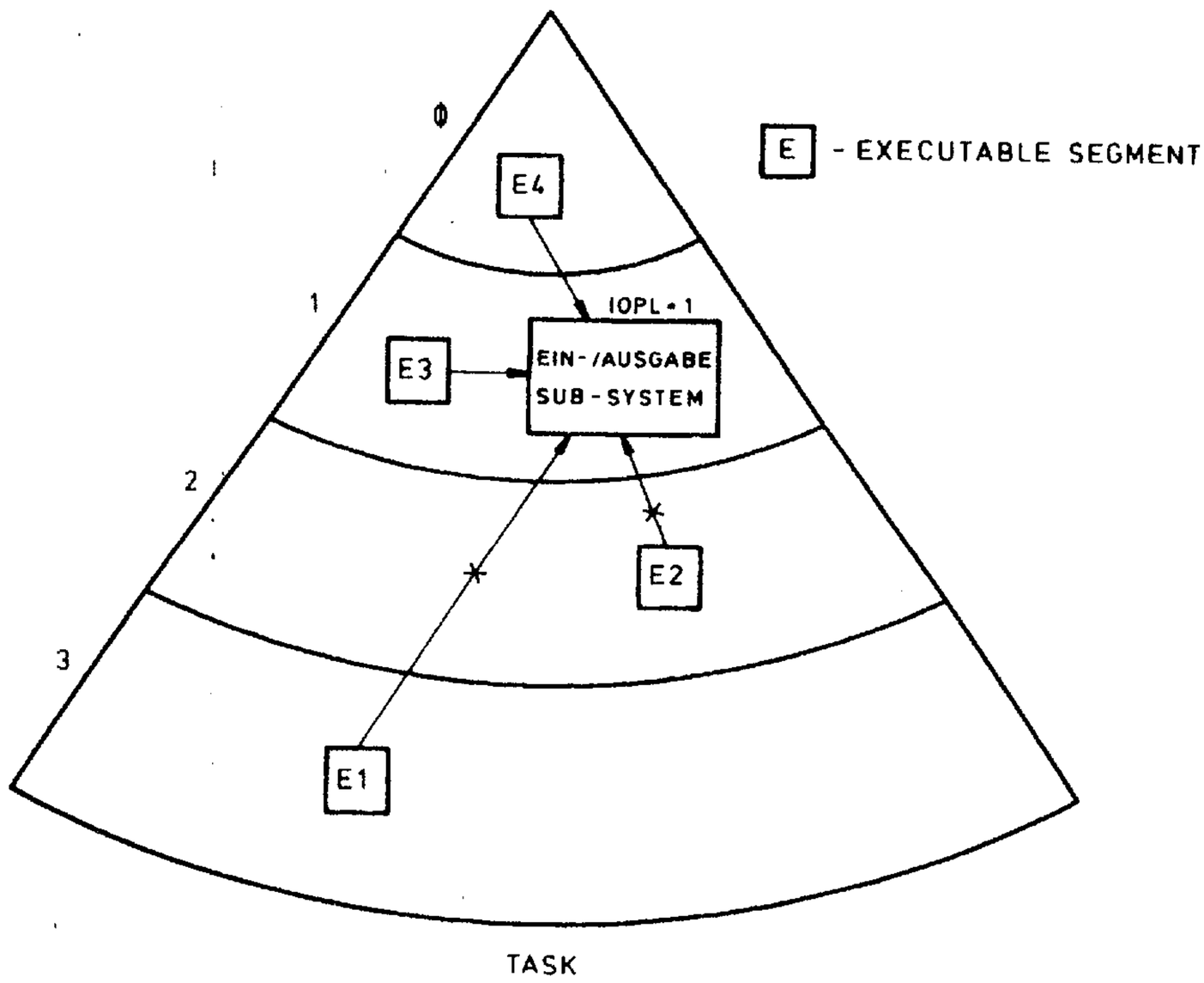
Sollte eine „I/O Permission Bit Map“ existieren, ist sie in diesem Fall ***nicht*** aktiv.

Nur wenn der I/O-Befehl in einer Privileg-Ebene ausgeführt wird, die numerisch ***größer*** als der IOPL-Wert ist (CPL > IOPL), konsultiert der 80486 als nächstes die I/O Permission Bit Map“ im augenblicklichen Task-Status-Segment und überprüft dabei die Bedingung

- BIT_MAP_OFFSET < TSS Limit.

Wenn diese Bedingung nicht erfüllt ist, existiert ***keine*** gültige „I/O Permission Bit Map“. Eine solche sogenannte „Null Permission Map“ ist mit einer Map identisch, die ausschließlich 1-Bits enthält und damit alle Port-Zugriffe verhindert. Daher führt bei dieser Situation die weitere Ausführung des I/O-Befehls zwangsläufig zur „Allgemeinen Schutzverletzung“.

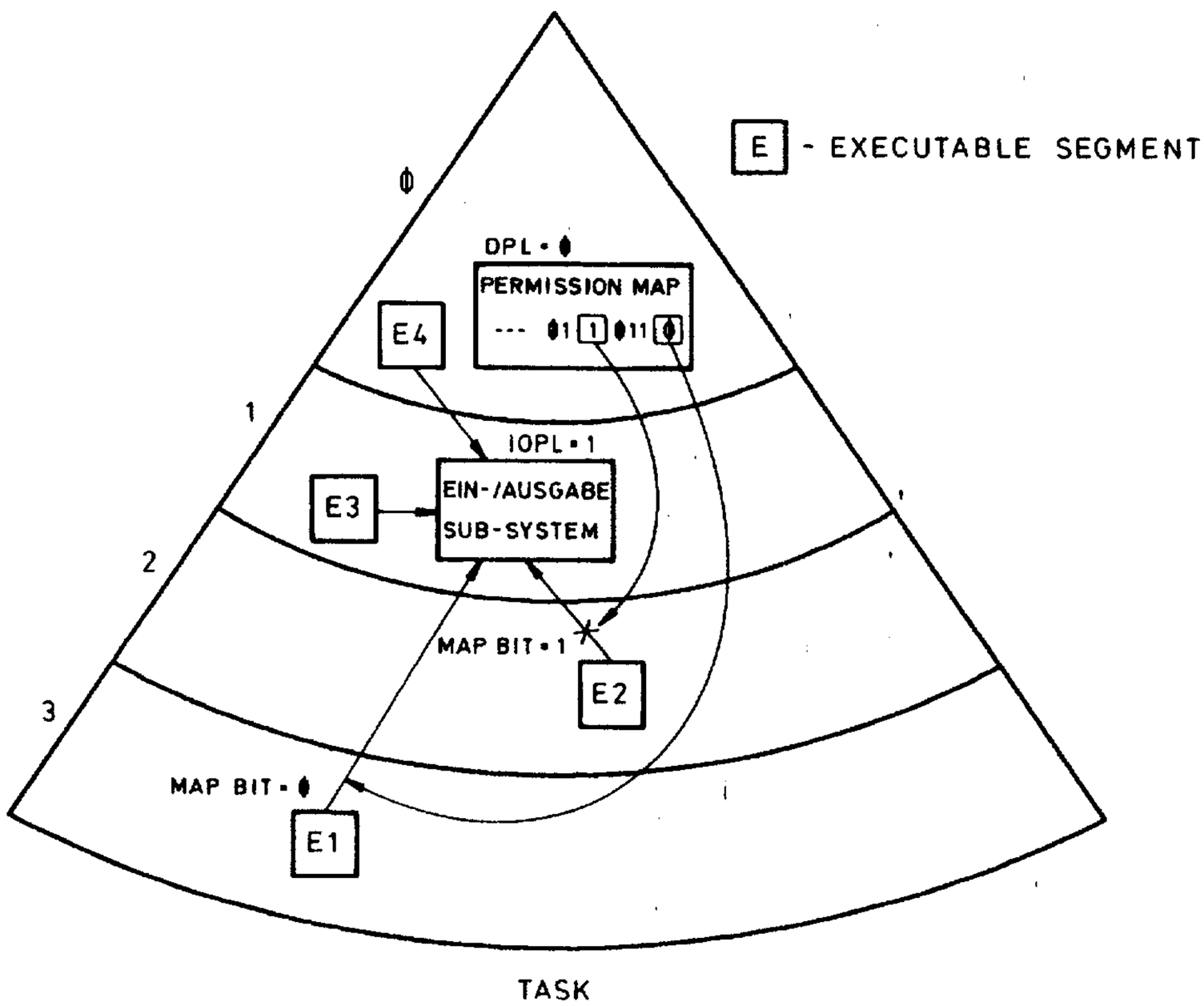
Bei der folgenden Software-Konfiguration ist vorausgesetzt, daß das Task-Status-Segment ***keine*** gültige „I/O Permission Bit Map“ enthält. Es illustriert gültige und ungültige Versuche, das Ein-/Ausgabe-Subsystem in der Privileg-Ebene 1 (IOPL = 01) durch „I/O-sensitive Befehle“ wie IN, INS, OUT oder OUTS zu erreichen.



Wie zu erkennen ist, können die „Executable“ Segmente ***E1*** und ***E2*** das Ein-/Ausgabe-Subsystem ***nicht*** benützen, während dies für ***E3*** und ***E4 möglich*** ist.

Wenn ein I/O-Befehl in einer höheren Privileg-Ebene als IOPL (CPL > IOPL) ausgeführt wird und dabei die Bedingung BIT_MAP_OFFSET < TSS Limit erfüllt ist, existiert eine ***gültige*** „I/O Permission Bit Map“. In diesem Fall kann die Ausführung des I/O-Befehls fortgesetzt werden. Dabei wird ein 0-Bit in der „Permission Map“ den Zugriff auf die korrespondierende I/O-Adresse erlauben, während ein 1-Bit den Zugriff auf den I/O-Port wegen einer „Allgemeinen Schutzverletzung“ verhindert.

Bei der folgenden Software-Konfiguration ist vorausgesetzt, daß das Task-Status-Segment eine gültige „I/O Permission Bit Map“ enthält. Es illustriert gültige und ungültige Versuche, das Ein-/Ausgabe-Subsystem in der Privileg-Ebene 1 (IOPL = 01) durch „I/O-sensitive“ Befehle wie IN, INS, OUT oder OUTS zu erreichen.



Es ist zu erkennen, daß die „Executabale“-Segmente ***E3*** und ***E4*** das Ein-/Ausgabe-Subsystem uneingeschränkt benützen können. Das ist deswegen so, weil bei CPL > IOPL die „I/O Permission Bit Map“ nicht ***aktiv*** ist. Der Zugriff auf das I/O-System durch die weniger privilegierten Segmente ***E1*** und ***E2*** wird nun über die „Permission Map“ gesteuert. Hält sich diese Map in der Privileg-Ebene 0 auf, ist nur das Betriebssystem, in der Lage, I/O-Zugriffe zu erlauben oder zu verhindern. Im Beispiel kann E1 den I/O-Port benützen (Map-Bit = 0), während E2 den I/O-Port nicht benützen kann (Map-Bit = 1).

HINWEIS:
Die „I/O Permission Bit Map“ ist immer dann von Nutzen, wenn ein Realzeit-System, das Zugriffe auf spezielle I/O-Ports überwacht, keine eigene Treiber-Software hat.

7.2.2 Änderung der I/O-Privilegebene

Soll IOPL geändert werden, kann dies nur in der Privileg-Ebene 0, z. B. vom Betriebssystem, realisiert werden. Allerdings gibt es ***keinen*** Befehl, der IOPL direkt beeinflußt. Die einzigen Mechanismen, die die EFLAGS verändern können, sind:

- Ein „Task"-Wechsel,
- die POPF/POPFD-Befehle (Pop FLAGS/EFLAGS) oder
- die IRET/IRETD-Befehle.

Wenn die IRET/IRETD- oder POPF/POPFD-Befehle in einer Privileg-Ebene ausgeführt werden, die numerisch ***größer*** als Null ist, werden zwar die anderen Flags, ***nicht*** aber IOPL in das CPU-Flag-Register zurückgeschrieben. Sollte dies vorkommen, gibt der Prozessor ***keine*** Fehlermeldung aus.

Beim Task-Wechsel werden die Flags vom Task-Status-Segment der eintretenden Task geladen. Solange aber für dieses Task-Status-Segment kein Alias-Daten-Segment in einer weniger privilegierten Ebene verfügbar ist, kann nur das Betriebssystem IOPL ändern.

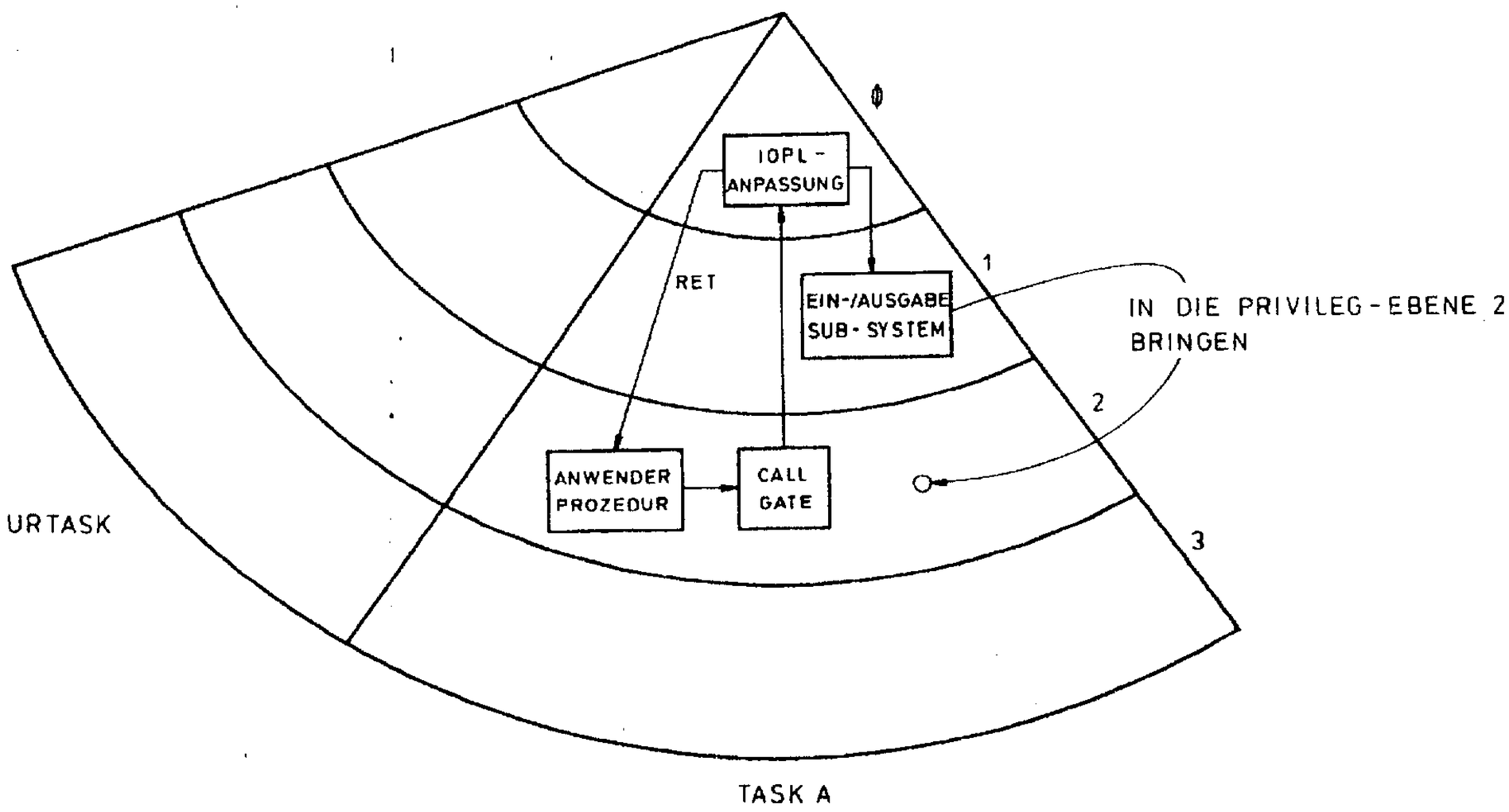
Ein 80486-System, das ***keine*** „I/O Permission Bit Map" benützt, regelt die I/O-Zugriffe ausschließlich über den IOPL-Schutzmechanismus.

Um in einem solchen System den maximalen Schutz zu erreichen, sollte sich das Ein/Ausgabe-Subsystem in einer Privileg-Ebene aufhalten, deren Kennummer

- ***größer*** als die des Betriebssystem-Kerns aber
- ***kleiner*** als die der Anwender-Prozeduren

ist.

Das folgende Bild illustriert eine solche Software-Konfiguration:



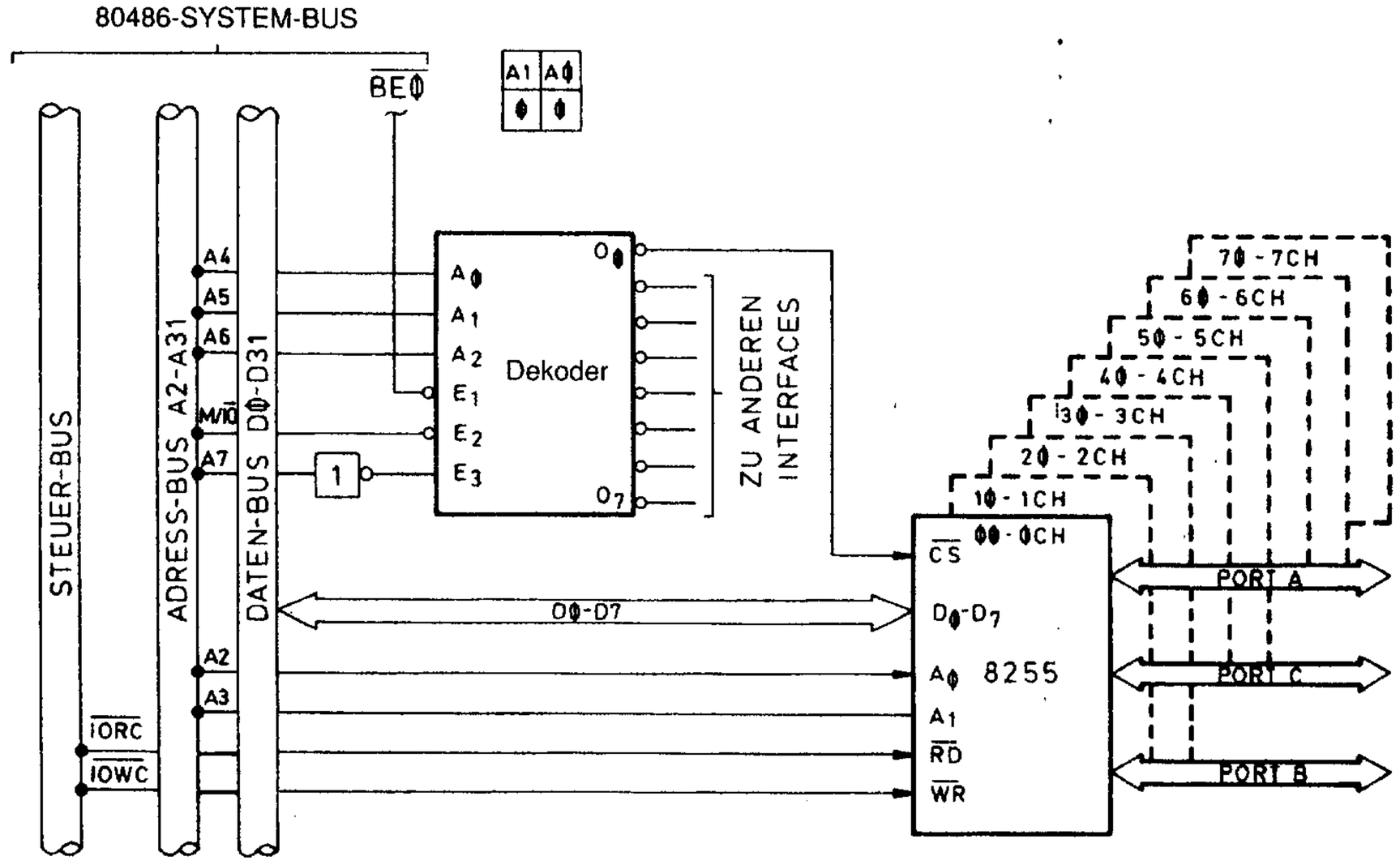
Es ist zu erkennen, daß sich die Anwender-Prozedur in der Privileg-Ebene 2 aufhält, während die Ein-/Ausgabe-Operationen in der Privileg-Ebene 1 ausgeführt werden (IOPL = 01).

Die Privileg-Anpassung der Ein-/Ausgabe-Komponenten übernimmt der Betriebssystem-Kern in der Privileg-Ebene 0.

Ist die Software in der angegebenen Weise konfiguriert, hat das Betriebssystem die Kontrolle über viele Ein-/Ausgabe-Operationen. Wenn nun weniger privilegierte Anwender-Prozeduren Ein-/Ausgabe-Funktionen benötigen, werden sie von IOPL gezwungen, das Betriebssystem aufzurufen.

7.3 Programmierbeispiel: IOPL-Anpassung

Das folgende ASM386/486-Programm-Fragment benützt die unten angegebene Hardware-Konfiguration un die oben beschriebene Software-Struktur.
Die Hardware enthält den parallelen Ein-/Ausgabe-Baustein 8255 mit der Basis-Adresse 0000H.



Das Programm-Modul enthält in der Privileg-Ebene 2 die FAR-Prozedur ANWENDER, in der zunächst mit CALL CALL_GATE die FAR-Prozedur IOPL_ANPASSUNG in der Privileg-Ebene 0 aufgerufen wird.

Diese Prozedur korrigiert mit

- PUSHFD
- POP EAX
- XOR AX,0011000000000000B
- PUSH EAX

das IOPL-Feld und bringt anschließend mit

- POPFD

den Ein-/Ausgabe-Baustein 8255 in die Privileg-Ebene 2.

Danach erfolgt mit RET der Rücksprung in die ANWENDER-Prozedur, die nun mit

- OUT DX,AL

auf das Betriebsarten-Register des 8255 (Adresse = 000CH) zugreifen kann (CPL ⩽ IOPL = 2).

```
LOC      OBJ                LINE    SOURCE

                             1          NAME        B18
                             2
                             3      EXTRN CALL_GATE:FAR
                             4
--------                     5      TASKA_CODE2     SEGMENT EO USE32      ;Nur ausfuehrbares Segment mit
                             6                                            ;Basis-Adresse=1000H und DPL=2
                             7                                            ;(Festlegung durch BLD386/486)
00000000                     8      ANWENDER        PROC FAR
                             9              ;
                            10              ;
00000000 9A00000000----E    11          CALL        CALL_GATE            ;Aufruf der Prozedur IOPL_ANPAS-
SUNG
                            12                                           ;ueber einen Call-Gate-Deskriptor
0000007  66BA0C00           13          MOV         DX,000CH             ;DX=Adresse des 8255-Betriebsarten-
                            14                                           ;Registers
0000000B B082               15          MOV         AL,82H               ;AL=Steuerwort; Port A=Ausgabe
0000000D EE                 16          OUT         DX,AL                ;Betriebsarten-Wort laden
                            17              ;
                            18              ;
0000000E                    19      ANWENDER        ENDP
                            20
--------                    21      TASKA_CODE_2    ENDS
***WARNING #377 IN 21, (PASS 2) SEGMENT CONTAINS PRIVILEGED INSTRUCTION(S)
                            22
--------                    23      TASKA_CODE_0    SEGMENT EO USE32      ;Nur ausfuehrbares Segment mit
                            24                                           ;Basis-Adresse=2000H und DPL=0
                            25                                           ;(Festlegung durch BLD386/486)
00000000                    26      IOPL_ANPASSUNG PROC FAR
                            27
00000000 9C                 28          PUSHFD                           ;Augenblickliche EFLAGS mit
                            29                                           ;IOPL=01 retten
00000001 58                 30          POP         EAX                  ;(EAX)=EFLAGS
00000002 66350030           31          XOR         AX,0011000000000000B ;Das IOPL-Feld bekommt den Wert 10
                            32                                           ;(I/O-Privileg=2)
00000006 50                 33          PUSH        EAX                  ;Neue EFLAGS retten
00000007 9D                 34          POPFD                            ;Die Privileg-Ebene des 8255 ist 2
00000008 CB                 35          RET                              ;Zurueck in die ANWENDER-Prozedur
                            36
00000009                    37      IOPL_ANPASSUNG ENDP
                            38
--------                    39      TASKA_CODE_0    ENDS
                            40
                            41          END

ASSEMBLY COMPLETE, 1 WARNING, NO ERRORS.
```

7.4 Interrupt-Flag und IOPL

Benützt ein Hardware-Interrupt, ein Interrupt-Befehl oder eine Ausnahme-Bedingung einen Interrupt-Gate-Deskriptor oder Trap-Gate-Deskriptor, darf sich der aufgerufene „Interrupt Handler" innerhalb der augenblicklichen Task entweder auf der gleichen oder einer numerisch kleineren Privileg-Ebene aufhalten. Dabei wird im Stack des „Interrupt Handlers" unter anderem der Flag-Status der unterbrochenen Prozedur gespeichert.

Wenn nun mit IRET/IRETD vom „Interrupt Handler" in die unterbrochene Prozedur zurückgekehrt wird, werden die Flags/EFLAGS vom Handler-Stack in das CPU-EFLAG-Register zurückgeschrieben.
Im Normalfall hat dann das IF-Flag (Interrupt Enable), wie vor dem Aufruf des „Interrupt Handlers" den Wert 1.

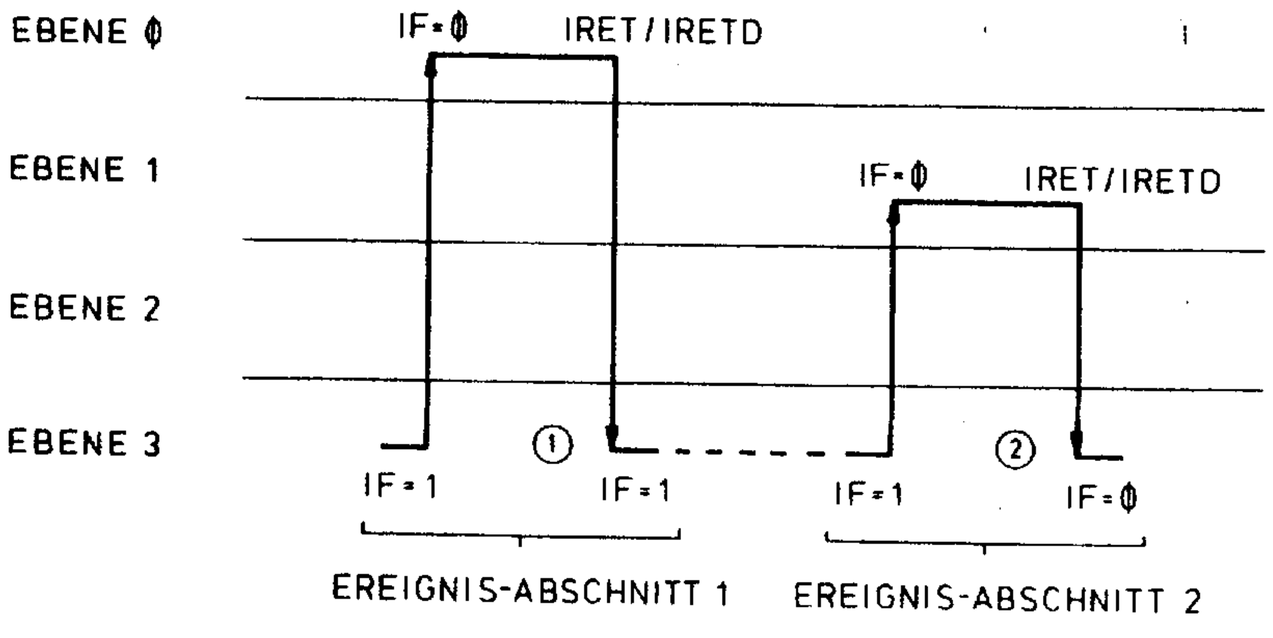
Wenn der „Interrupt Handler" über einen Interrupt-Gate-Deskriptor (***nicht*** Trap-Gate-Deskriptor) aufgerufen worden ist und irgendwann IRET/IRETD in einer Privileg-Ebene ausführt, deren Kennummer numerisch größer als IOPL ist, bleibt das IF-Flag gelöscht (IF = 0).

Das IF-Flag bekommt beim Verlassen eines solchen „Interrupt Handlers" nur dann wieder den Wert 1, wenn die Bedingung

- „Task" CPL ⩽ IOPL

erfüllt ist.

Das folgende Bild zeigt hierfür zwei Beispiele. Dabei ist für ***IOPL*** der ***Wert 0*** vorausgesetzt.



Im Ereignis-Abschnitt 1 wird eine Prozedur in der Privileg-Ebene 3 unterbrochen (IF = 1) und ein „Interrupt Handler" in der Privileg-Ebene 0 aufgerufen. Danach hat IF den Wert 0. Da die Task IRET/IRETD in der Privileg-Ebene 0 ausführt, ist die Bedingung Task CPL ⩽ IOPL ***erfüllt***, so daß nach seiner Ausführung IF den Wert 1 bekommt ①.

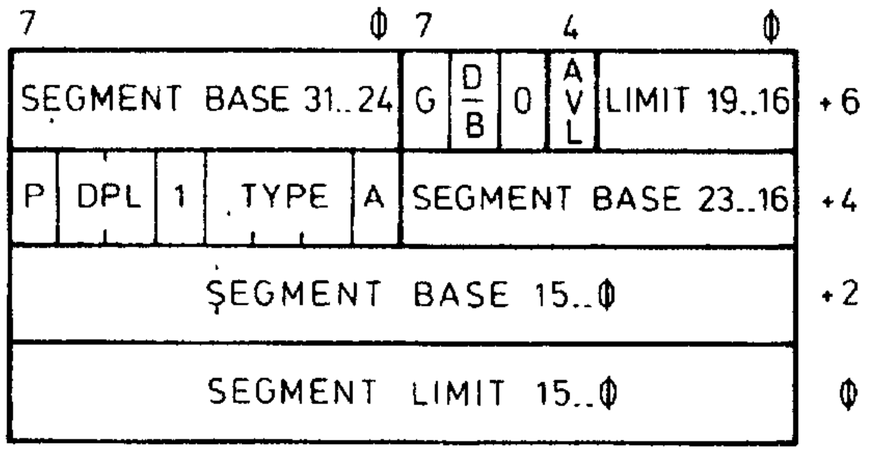
Im Ereignis-Abschnitt 2 wird ebenfalls eine Prozedur in der Privileg-Ebene 3 unterbrochen (IF = 1), aber ein „Interrupt Handler" in der Privileg-Ebene 1 aufgerufen. Danach hat IF den Wert 0. Da die Task IRET/IRETD in der Privileg-Ebene 1 ausführt, ist die Bedingung Task CPL ⩽ IOPL ***nicht*** erfüllt, so daß nach seiner Ausführung IF den Wert 0 behält ②.

7.5 Sperren von Segmenten und Pages bei direkten I/O-Operationen

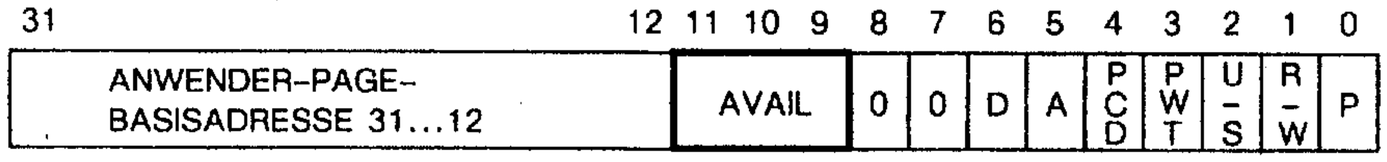
In einem 80486-System werden direkte I/O-Operationen typischerweise mit einem DMA-Controller durchgeführt. Das Betriebssystem muß dabei garantieren, daß die am I/O-Transfer beteiligten Segmente und Pages so lange ***nicht*** bewegt, gelöscht oder als nicht-präsent markiert werden, bis der Transfer beendet ist.

Zu diesem Zweck kann das Betriebssystem das Available-Bit (AVL) in den Segment-Deskriptoren benützen, um ein Segment als ***„gesperrt für I/O“*** zu kennzeichnen. Um eine Anwender-Page als „gesperrt für I/O“ zu kennzeichnen, können die drei Available-Bits (AVAIL) im zugehörigen Page-Deskriptor benützt werden. Die Positionen dieser Bits sind in den folgenden Deskriptor-Layouts zu sehen:

SEGMENT DESKRIPTOR:



'PAGE'-DESKRIPTOR:



Das Betriebssystem darf den Markierungs-Code im Available-Feld eines Segment- oder Page-Deskriptors erst dann ändern, wenn die I/O-Operation vollständig abgeschlossen ist. Im anderen Fall ist die Gefahr groß, daß es zu einem unvollständigen I/O-Transfer kommt.