11 DEBUG-Unterstützung

Der 80486 hat mehrere Funktionen, die den Debug-Prozeß vereinfachen und dabei die Single-Step- und Breakpoint-Ausnahmen der Vorgänger-Prozessoren noch verfügbar halten. Die wichtigste Debug-Unterstützung beim 80486 kommt aber von sogenannten ***Debug-Register.***

Diese Register erlauben die Definition von Breakpoints sowohl bei Datenzugriffen als auch bei der Ausführung von Befehlen. Sie eliminieren z. B. die Schwierigkeiten, die beim Schreiben eines Breakpoint-Befehls in ein Code-Segment auftreten. Bei einer solchen Aktion war in der Vergangenheit für das Code-Segment ein Alias-Daten-Segment erforderlich.

Durch die Debug-Register besteht sogar die Möglichkeit, Breakpoints in einem Code zu setzen, der sich im ROM aufhält oder der von verschiedenen Tasks benützt wird.

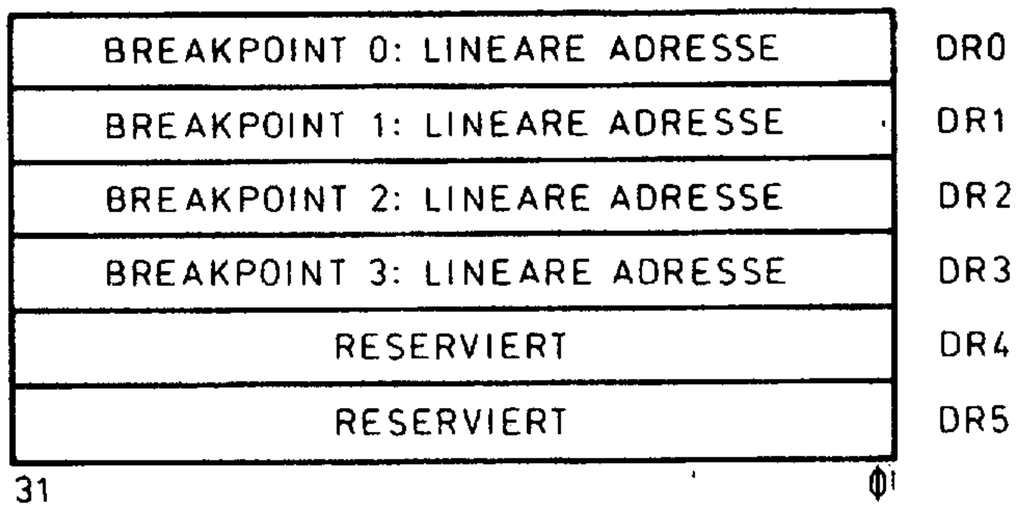
11.1 Die Breakpoint-Register DR0 ... DR5

Es existieren sechs Breakpoint-Register, wobei mit DR0 ... DR3 vier verschiedene Breakpoint-Adressen spezifiziert werden können. Wie das folgende Bild zeigt, müssen zu diesem Zweck passende 32-Bit lineare Operanden- oder Befehls-Adressen in diese Register geschrieben werden.

ZUR ERINNERUNG:

Eine lineare Adresse ist die Summe aus einer 32-Bit-Segment-Basis-Adresse und einer effektiven Adresse.

DEBUG - BREAKPOINT - REGISTER:

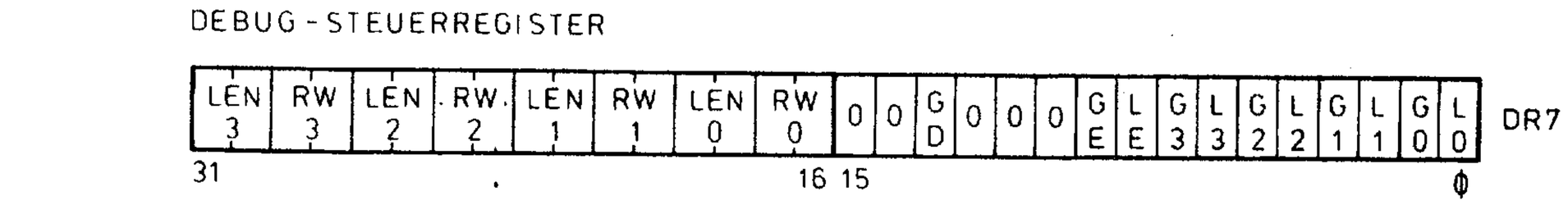


Die 80486-Hardware vergleicht kontinuierlich die linearen Breakpoint-Adressen in DR0 ... DR3 mit den linearen Adressen, die von der Software generiert werden.

Wenn der Paging-Mechanismus gesperrt ist, sind die linearen Adressen mit den physikalischen Adressen identisch. Wenn der Paging-Mechanismus freigegeben ist, werden die linearen Adressen in physikalische Adressen umgesetzt. Unabhängig davon, ob Paging freigegeben ist oder nicht, enthalten die Breakpoint-Register immer lineare Adressen.

11.2 Das Debug-Steuerregister DR7

Das Debug-Steuerregister erlaubt die Definition von Debug-Bedingungen und ihre selektive Freigabe oder Sperre. Das folgende Bild zeigt das Layout von DR7:



Wie zu sehen ist, enthält DR7 eine Reihe von Feldern, die mit den linearen Adressen in den Breakpoint-Registern DR0 ... DR3 korrespondieren.

11.2.1 RW0 ... RW3 (Abbruch-Ereignis)

Für jede der vier Breakpoint-Adressen in DR0 ... DR3 existiert ein 2-Bit RW-Feld. Dieses Feld spezifiziert, welches Ereignis einen Programm-Abbruch aktivieren soll. Wie der Prozessor den Code in diesen Feldern interpretiert, zeigt die folgende Tabelle:

RW-Code	Abbruch-Ereignis
00	Nur bei einer Befehlsausführung
01	Nur beim Schreiben von Daten
10	Undefiniert
11	Nur beim Schreiben oder Lesen von Daten. Nicht beim Holen von Befehlen.

Der RW-Code 00 wird benützt, um einen Breakpoint bei der Ausführung eines Befehls zu setzen. Dabei werden solche Breakpoints als ***Faults*** interpretiert. Dies bedeutet, daß der Programm-Abbruch ***vor*** der Ausführung des Befehls erfolgt.

Die RW-Codes 01 oder 11 werden benützt, um Breakpoints bei schreibenden oder lesenden Datenzugriffen zu setzen. Solche Breakpoints werden als ***Traps*** interpretiert. Dies bedeutet, daß der Programm-Abbruch ***nach*** dem Daten-Transfer erfolgt.

11.2.2 LENO ... LEN3 (Länge des Breakpoint-Feldes)

Für jede der vier Breakpoint-Adressen in DR0 ... DR3 existiert ein 2-Bit LEN-Feld. Dieses Feld bestimmt, welche der niederwertigen Bits in einer Breakpoint-Adresse benützt werden, um das Breakpoint-Ereignis zu entdecken. Mit anderen Worten, das LEN-Feld definiert zusammen mit der zugehörigen Breakpoint-Adresse in DR0 ... DR3 den ***Bereich*** der sequentiellen Byte-Adressen für einen Daten-Breakpoint. Dieser Byte-Bereich wirkt als sogenanntes ***Breakpoint-Feld.*** Dabei erlaubt LEN die Spezifikation von 1-Byte-, 2-Byte- oder 4-Byte-Breakpoint-Feldern. Dies bedeutet, daß die Breakpoint-Adresse auf

ein ganzes Feld von Byte-Einheiten zeigt, die bei einem Datenzugriff entweder gelesen oder geschrieben werden.

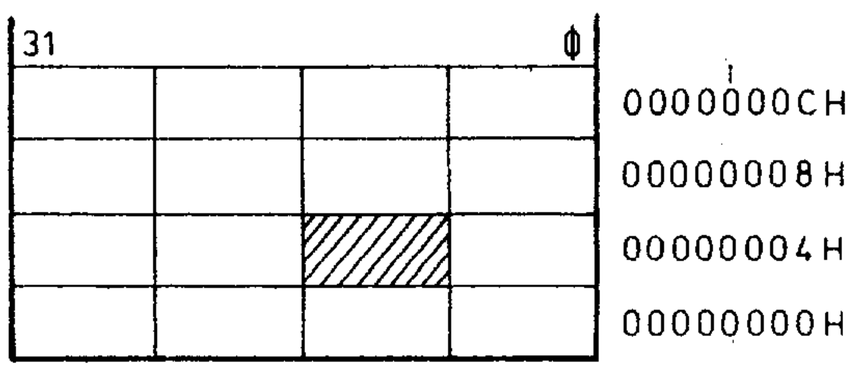
Immer dann, wenn die Software bei einem Datenzugriff Byte-Adressen liefert, die in ein solches Breakpoint-Feld fallen, triggert der Prozessor einen Programm-Abbruch. Wie der 80486 den Code im LEN-Feld interpretiert, zeigt die folgende Tabelle:

LEN-Code	Größe des Breakpoint-Feldes	Beanspruchung der niederwertigen Bits in der Breakpoint-Adresse
00	1 Daten-Byte	Im Breakpoint-Adreß-Register werden alle 32 Adreß-Bits benützt, um ein Breakpoint-Feld mit einem einzigen Daten-Byte zu spezifizieren.
01	2 Daten-Bytes	Im Breakpoint-Adreß-Register werden nur A1 ... A31 (nicht A0) benützt, um ein Breakpoint-Feld mit zwei Daten-Bytes zu spezifizieren.
10	Undefiniert	
11	4 Daten-Bytes	Im Breakpoint-Adreß-Register werden nur A2 ... A31 (nicht A0 und A1) benützt, um ein Breakpoint-Feld mit vier Daten-Bytes zu spezifizieren.

Damit bei Daten-Zugriffen der erwartete Programm-Abbruch eintritt, müssen die Breakpoint-Felder an den richtigen Breakpoint-Adressen ausgerichtet sein. Diese Anforderungen erzwingt der Prozessor, indem er die niederwertigen Adreß-Bits in den Breakpoint-Registern DR0 ... DR3 maskiert. Dabei werden 1-Byte-Breakpoint-Felder wegen LEN = 00 an beliebigen Adreßgrenzen (A0 = X), 2-Bytes-Breakpoint-Felder wegen LEN = 01 an Word-Adreßgrenzen (A0 = 0) und 4-Byte-Breakpoint-Felder wegen LEN = 11 an Doubleword-Adreßgrenzen (A1/A0 = 00) ausgerichtet.

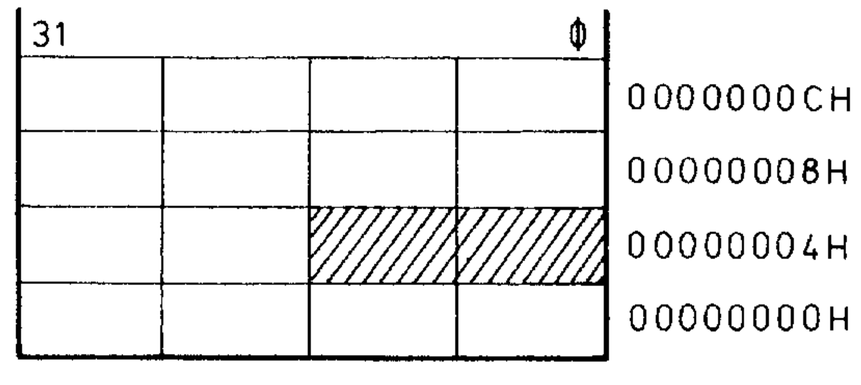
BEISPIEL:
Es sei vorausgesetzt, daß das Breakpoint-Register DR2 die lineare Adresse 00000005H enthält.

1. **LEN2 = 00** (1-Byte-Breakpoint-Feld)



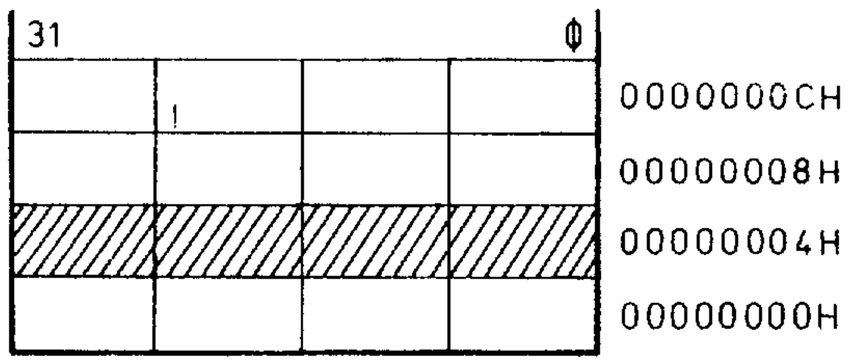
Da der Prozessor für den Adreßvergleich alle 32 Adreß-Bits in DR2 benützt, beginnt das 1-Byte-Breakpoint-Feld bei der linearen Adresse 00000005H.

2. LEN2 = 01 (2-Byte-Breakpoint-Feld)



Da der Prozessor für den Adreßvergleich nur A31 ... A1 (A0 = X) in DR2 benützt, beginnt das 2-Byte-Breakpoint-Feld bei der linearen Adresse 00000004H (AO = 0) und endet bei 00000005H (A0 = 1).

3. LEN2 = 11 (4-Byte-Breakpoint-Feld)



Da der Prozessor für den Adreßvergleich nur A31 ... A2 (A1/A0 = XX) in DR2 benützt, beginnt das 4-Byte-Breakpoint-Feld bei der linearen Adresse 00000004H (A1/A0 = 00), umfaßt 00000005H (A1/A0 = 01), 00000006H (A1/A0 = 10) und endet bei 00000007H (A1/A0 = 11).

Wenn bei einem Datenzugriff die von der Software generierten linearen Byte-Adressen ***komplett*** oder ***teilweise*** mit den Adressen der Breakpoint-Felder übereinstimmen, erkennt der Prozessor die sogenannte ***Debug-Ausnahme*** (Interrupt 1), die zum Aufruf des zugehörigen Ausnahme-Handlers führt.

Die folgenden Tabellen zeigen einige Beispiele für Speicher-Referenzen, bei denen der Prozessor-Debug-Ausnahmen erkennt bzw. nicht erkennt. Die angegebenen Speicher-Referenzen (lineare Adressen der Software) beziehen sich dabei auf die folgenden Debug-Bedingungen in DR0 ... DR3:

		Breakpoint-Adresse (hexadezimal)	Byte-Länge des Breakpoint-Feldes
Register-Inhalte	DR0	0A0001	1 (LEN0 = 00)
	DR1	0A0002	1 (LEN1 = 00)
	DR2	0B0002	2 (LEN2 = 01)
	DR3	0C0000	4 (LEN3 = 11)

1. Speicher-Referenzen, die zur Debug-Ausnahme führen

Lineare Adresse der Software (hexadezimal)	Länge des Operanden
0A0001	1
0A0002	1
0A0001	2
0A0002	2
0B0002	2
0B0001	4
0C0000	4
0C0001	2
0C0003	1

2. Speicher-Referenzen, die zu keiner Debug-Ausnahme führen

Lineare Adresse der Software (hexadezimal)	Länge des Operanden
0A0000	1
0A0003	4
0B0000	2
0C0004	4

Wenn ein Programm-Abbruch ***vor*** einem Befehl erfolgen soll, ***muß*** in den Breakpoint-Registern DR0 ... DR3 der RW-Code = 00 sein und die Breakpoint-Felder müssen eine Länge von 1 Byte haben (LEN = 00).

Dies bedeutet, daß der Prozessor die Breakpoint-Adresse eines Befehls nur dann erkennt, wenn sie auf das erste Byte des Befehls zeigt. Sollte der Befehl Prefixes haben, muß die Breakpoint-Adresse zur ersten Prefix zeigen.

11.2.3 G0 ... G3 und L0 ... L3 (Breakpoint-Freigabe, global und lokal)

Die niederwertigen 8 Bits im Register DR7 geben selektiv folgende Breakpoint-Bedingungen frei:

- die linearen Adressen (in DR0 ... DR3),
- die Länge der Breakpoint-Felder (LEN0 ... LEN3 in DR7),
- die Abbruch-Ereignisse (RW0 ... RW3 in DR7).

Dabei gibt es zwei Ebenen der Freigabe:

- die lokale Ebene (L0 ... L3),
- die globale Ebene (G0 ... G3).

Dies hat folgenden Grund: Wenn der Paging-Mechanismus eingeschaltet ist, können in den verschiedenen Tasks die linearen Adressen auf unterschiedliche physikalische Adressen abgebildet werden. Sollte dies der Fall sein, kann die Breakpoint-Adresse in der einen

Task relevant sein, in der anderen aber nicht. Eine Breakpoint-Adresse, die für ***alle*** Tasks gelten soll, wird über G0 ... G3 gesteuert (G0 ... G3 = 1). Eine Breakpoint-Adresse, die nur für eine bestimmte Task gelten soll, wird über L0 ... L3 gesteuert (L0 ... L3 = 1).

Die lokalen Freigabe-Bits werden bei einem Task-Wechsel automatisch gelöscht. Dadurch können unerwünschte Breakpoint-Bedingungen in der neuen Task verhindert werden. Im Gegensatz dazu werden die globalen Freigabe-Bits bei einem Task-Wechsel nicht gelöscht. Sie können also für Abbruch-Bedingungen benützt werden, die für alle Tasks global sind.

11.2.4 GE und LE (Exakter Daten-Breakpoint, global und lokal)

Wenn GE oder LE gesetzt sind, verlangsamt der Prozessor die Befehlsausführung. Bevor die „Execution Unit" des 80486 die Ausführung des nächsten Befehls beginnt, wartet sie auf den Abschluß des augenblicklichen Daten-Transfers. Dies hat den Vorteil, daß bei einem Daten-Breakpoint die Programm-Unterbrechung unmittelbar nach dem Daten-Transfer erfolgt. Sollten GE oder LE gelöscht sein, wird bei einem Daten-Breakpoint die Programm-Unterbrechung erst einige Befehle später ausgelöst. Es ist daher empfehlenswert, GE oder LE zu benützen.

Wenn der 80486 einen Task-Wechsel einleitet, wird LE gelöscht. Dadurch wird verhindert, daß die neue Task auf einen exakten Daten-Breakpoint trifft. LE unterstützt daher einen schnellen Task-Wechsel.

Das GE-Bit bleibt während eines Task-Wechsels unbeeinflußt. Es unterstützt exakte Daten-Breakpoints während der Ausführung ***aller*** Tasks.

> **HINWEIS:**
> Befehl-Breakpoints führen immer zu exakten Programm-Unterbrechungen, unabhängig davon, ob ein exakter Daten-Breakpoint definiert ist oder nicht.

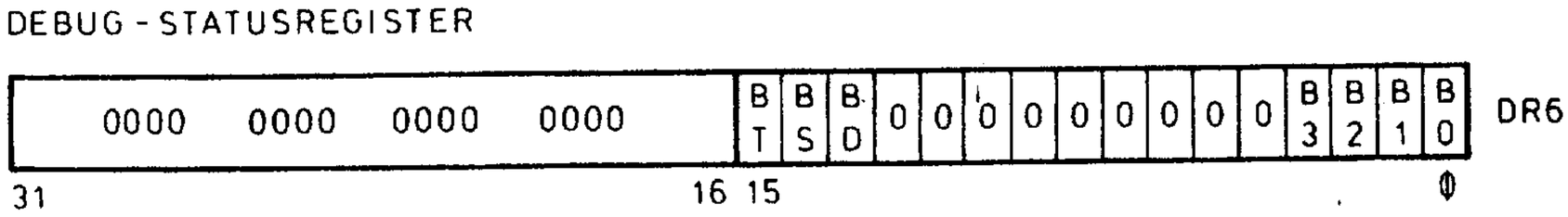
11.2.5 GD (Zugriff auf Debug-Register steuern)

Der Zugriff auf die Debug-Register ist nur im „Real Mode" oder in der Privileg-Ebene 0 im „Protected Mode" erlaubt. Ist GD gesetzt (GD = 1), kann anschließend auf die Debug-Register ***nicht*** mehr zugegriffen werden. Dadurch wird garantiert, daß ein Software-Debugger die volle Kontrolle über die Debug-Register behält. Der Versuch, bei GD = 1 irgendein Debug-Register zu manipulieren, führt zur Debug-Ausnahme (Interrupt 1).

Nach dem Aufruf des Ausnahme-Handlers wird GD automatisch gelöscht. Der Handler kann nun den freien Zugriff auf die Debug-Register erlauben (GD = 0) oder verhindern (GD = 1).

11.3 Das Debug-Status-Register DR6

Das Debug-Status-Register erlaubt dem Ausnahme-Handler die Bestimmung der aufgetretenen Debug-Bedingungen. Das Layout dieses Registers zeigt das folgende Bild:



11.3.1 B0 ... B3 (Debug-Fault/Trap wegen Breakpoints 0 ... 3)

Die vier Breakpoint-Indikator-Flags B0 ... B3 korrespondieren mit den Breakpoint-Registern DR0 ... DR3. Dabei wird Bn ***vor*** dem Aufruf des Debug-Handlers gesetzt, wenn der Prozessor die durch DRn, LENn und RWn definierte und freigegebene Breakpoint-Bedingung entdeckt.

Der Debug-Handler behandelt die Ausnahme als ***„Fault“*** (Befehl wiederholbar), wenn sie durch einen ***Befehls***-Breakpoint verursacht worden ist, und als ***„Trap“*** (Befehl nicht wiederholbar), wenn sie durch einen ***Daten***-Breakpoint verursacht worden ist.

Ein Bn-Flag wird immer dann gesetzt, wenn die Hardware auf eine freigegebene Breakpoint-Bedingung trifft. Wenn ein solcher Treffer bei mindestens einem freigegebenen Breakpoint n entdeckt worden ist, setzt die Hardware alle Bn-Bits, die mit den Breakpoint-Bedingungen im Augenblick korrespondieren. Dies geschieht unabhängig davon, ob die Bedingungen freigegeben sind oder nicht. Da also der Ausnahme-Handler unter Umständen mehrere gesetzte Bn-Bits erkennen kann, sind für ihn nur diejenigen Bn-Bits gültig, deren korrespondierende Breakpoints mit Ln oder Gn freigegeben sind.

11.3.2 BD (Debug-Fault, wenn GD gesetzt ist)

Dieses Bit wird gesetzt (BD = 1), wenn bei GD = 1 in DR7 der Debug-Handler durch einen Zugriffsversuch auf eines der Debug-Register aufgerufen worden ist. Wenn ein solches Ereignis vorkommt, wird GD nach dem Aufruf des Debug-Handlers automatisch gelöscht. Der Handler kann nun entscheiden, ob er den freien Zugriff auf die Debug-Register erlaubt (GD = 0) oder nicht (GD = 1).

11.3.3 BS (Debug-Trap wegen Single-Step)

Wenn der Prozessor nach der Ausführung eines Befehls entdeckt, daß das TF-Bit (Trap Flag) im EFLAG-Register gesetzt war (TE = 1), setzt er das BS-Bit in DR6 und ruft den sogenannten Single-Step-Ausnahme-Handler auf. Der Handler behandelt diese Ausnahme als „Trap“ (Befehl nicht wiederholbar).

Die Single-Step-Ausnahme wird ***nicht*** generiert, wenn ein Befehl das TF-Bit setzt. Sollte z. B. POPF benützt werden, um das TF-Bit zu setzen, kommt die Single-Step-Ausnahme

erst bei dem Befehl vor, der dem POPF-Befehl unmittelbar folgt. Der Prozessor löscht TF ***vor*** dem Aufruf des Handlers. Ist TF im EFLAG-Feld eines Task-Status-Segments während eines Task-Wechsels gesetzt (TF = 1), wird die Ausnahme nach der Ausführung des ersten Befehls in der neuen Task generiert.

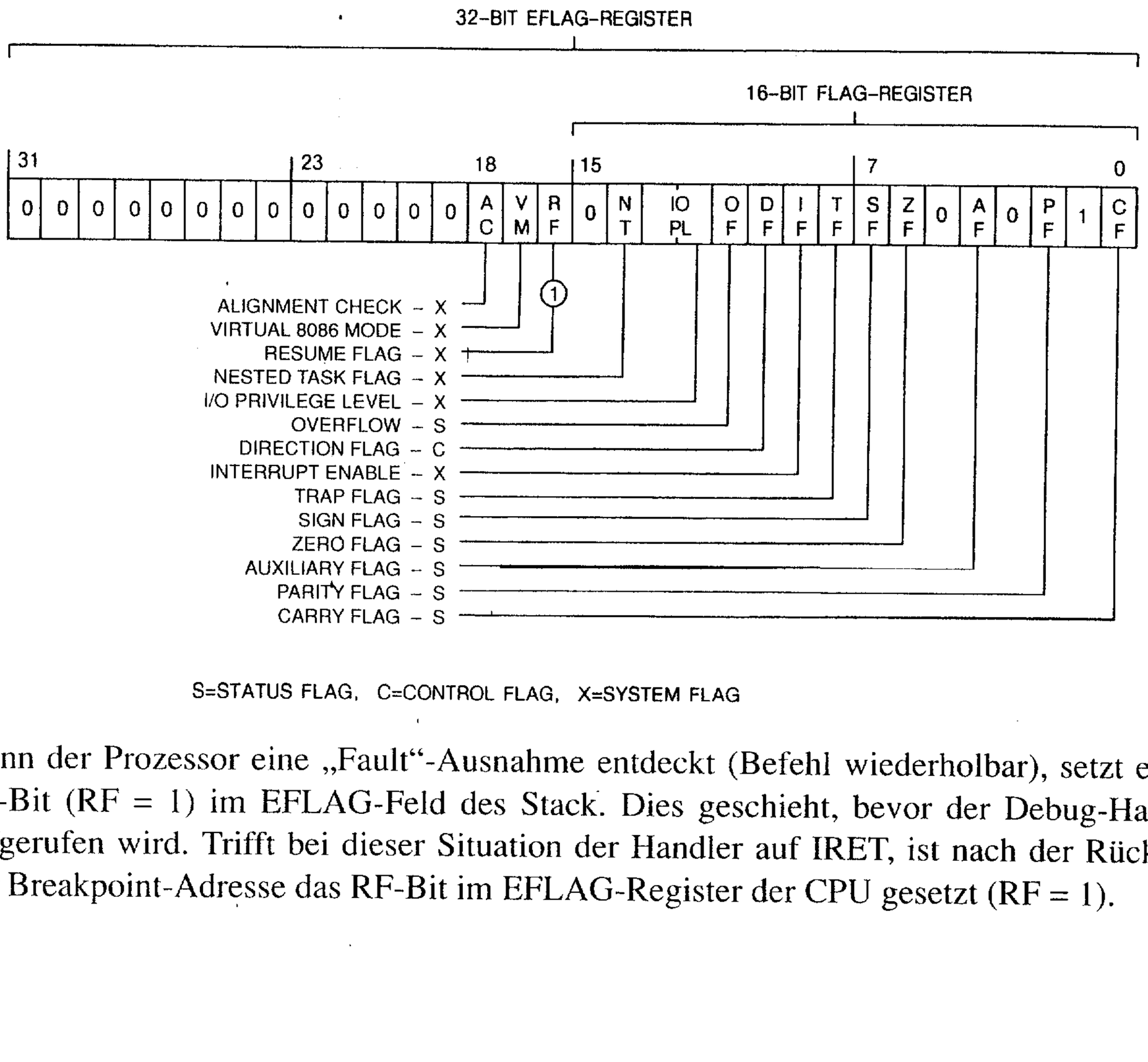
Das TF-Flag wird normalerweise bei einem Privileg-Wechsel innerhalb einer Task ***nicht*** gelöscht. Davon ausgenommen sind jedoch Privileg-Wechsel, die durch (INT n)-Befehle ausgelöst werden. Hier wird das TF-Flag gelöscht (TF = 0). Aus diesem Grund muß der Software-Debugger INT n oder INTO erkennen und emulieren.

11.3.4 BT (Debug-Trap wegen Task-Switch)

Die Debug-Ausnahme kommt auch vor, wenn nach einem Task-Wechsel im neuen Task-Status-Segment das T-Bit gesetzt ist. Dabei erfolgt der Wechsel in die neue Task zunächst normal. Bevor jedoch der erste Befehl in der neuen Task ausgeführt wird, erfolgt der Aufruf des Debug-Handlers. Damit der Handler diese Ausnahme feststellen kann, setzt der Prozessor das BT-Bit (BT = 1).

11.4 Benützung des Resume-Flags RF

Das RF-Bit im EFLAG-Register der CPU wird in Verbindung mit den Debug-Breakpoints benützt. Die Position dieses Flags ist im EFLAG-Layout zu sehen ①:



Wenn der Prozessor eine „Fault"-Ausnahme entdeckt (Befehl wiederholbar), setzt er das RF-Bit (RF = 1) im EFLAG-Feld des Stack. Dies geschieht, bevor der Debug-Handler aufgerufen wird. Trifft bei dieser Situation der Handler auf IRET, ist nach der Rückkehr zur Breakpoint-Adresse das RF-Bit im EFLAG-Register der CPU gesetzt (RF = 1).

Wegen RF = 1 kann nun der Prozessor den Befehl bei der gleichen Breakpoint-Bedingung wiederholen, ***ohne*** daß eine weitere Ausnahme generiert wird.

Bei der Wiederholung des Befehls kann es vorkommen, daß der Prozessor auf andere „Faults" trifft. Der Befehl ist aber mit RF = 1 solange wiederholbar, bis er erfolgreich ausgeführt worden ist. Ist dies der Fall, löscht der Prozessor das RF-Bit automatisch (RF = 0).

Das RF-Bit wird nach Ausführung der folgenden Befehle ***nicht*** gelöscht:

- IRET, POPF und
- JMP, CALL oder INT n,

  wenn letztere einen Task-Wechsel verursachen.

11.5 Erkennen von Debug-Bedingungen

Der Debug-Handler ist normalerweise ein selbständiger Debugger oder ein Teil eines kompletten Debug-Systems. Da der Prozessor einen Interrupt 1 bei vielen Breakpoint-Bedingungen generiert, kann der Debugger DR6 und DR7 konsultieren, um den Typ der Bedingung festzustellen.

Abhängig von den Breakpoint-Bedingungen liefert der Prozessor eine Kombination von Flags, die der Debugger testen muß, wenn er den Typ der Breakpoint-Bedingung bestimmen will.

Die folgende Tabelle zeigt den Zusammenhang zwischen den Breakpoint-Bedingungen und den zu testenden Flags:

Zu testende Flags	Bedingung
BS = 1	Single-Step
B0 = 1 AND (GE0 = 1 OR LE0 = 1)	Breakpoint DR0, LEN0, R/W0
B1 = 1 AND (GE1 = 1 OR LE1 = 1)	Breakpoint DR1, LEN1, R/W1
B2 = 1 AND (GE2 = 1 OR LE2 = 1)	Breakpoint DR2, LEN2, R/W2
B3 = 1 AND (GE3 = 1 OR LE3 = 1)	Breakpoint DR3, LEN3, R/W3
BD = 1	Debug-Register
BT = 1	Task-Wechsel

11.6 Zugriff auf die Debug-Register

Damit die Debug-Register DR0 ... DR7 gelesen und beschrieben werden können, stehen folgende MOV-Spezial-Befehle zur Verfügung:

```
• MOV   r32,DR0/DR1/DR2/DR3     ; Lade Debug-Register DR0 ... DR3 in
                                ; ein 32-Bit allgemeines Register
• MOV   r32,DR6/DR7             ; Lade Debug-Register DR6, DR7 in ein
                                ; 32-Bit allgemeines Register
• MOV   DR0/DR1/DR2/DR3,r32     ; Lade ein 32-Bit allgemeines Register
                                ; ins Debug-Register DR0 ... DR3
```

```
•   MOV     DR6/DR7,r32          ; Lade ein 32-Bit allgemeines Register
                                 ; ins Debug-Register DR6, DR7
```

11.7 Programmierbeispiel:

Definition von Abbruch-Bedingungen

Das folgende ASM386/486-Programm-Fragment enthält die Definition von vier Breakpoint-Bedingungen:

```
LOC       OBJ       LINE      SOURCE
                      1                          NAME    BEISPIEL 23
                      2
                  #   3      DR7_FELDER RECORD   LEW3:   2,    ;Laenge des Breakpoint-Feldes fuer DR3
                      4      &                   RW3:    2,    ;Abbruch-Ereignis fuer DR3
                      5      &                   LEN2:   2,    ;Laenge des Breakpoint-Feldes fuer DR2
                      6      &                   RW2:    2,    ;Abbruch-Ereignis fuer DR2
                      7      &                   LEN1:   2,    ;Laenge des Breakpoint-Feldes fuer DR1
                      8      &                   RW1:    2,    ;Abbruch-Ereignis fuer DR1
                      9      &                   LEN0:   2,    ;Laenge des Breakpoint-Feldes fuer DR0
                     10      &                   RW0:    2,    ;Abbruch-Ereignis fuer DR0
                     11      &                   XX:     2,    ;Undefiniert
                     12      &                   GD:     1,    ;Zugriff auf die Debug-Register steuern
                     13      &                   XXX:    3,    ;Undefiniert
                     14      &                   GEX:    1,    ;Exakter Daten-Breakpoint global
                     15      &                   LEX:    1,    ;Exakter Daten-Breakpoint lokal
                     16      &                   G3:     1,    ;Globale Breakpoint-Freigabe fuer DR3
                     17      &                   L3:     1,    ;Lokale Breakpoint-Freigabe fuer DR3
                     18      &                   G2:     1,    ;Globale Breakpoint-Freigabe fuer DR2
                     19      &                   L2:     1,    ;Lokale Breakpoint-Freigabe fuer DR2
                     20      &                   G1:     1,    ;Globale Breakpoint-Freigabe fuer DR1
                     21      &                   L1:     1,    ;Lokale Breakpoint-Freigabe fuer DR1
                     22      &                   G0:     1,    ;Globale Breakpoint-Freigabe fuer DR0
                     23      &                   L0:     1     ;Lokale Breakpoint-Freigabe fuer DR0
                     24
000A0001             25      DR0_BREAKPOINT_ADRESSE      EQU 000A0001H  ;Breakpoint-Adresse im DR0-Register
000A0002             26      DR1_BREAKPOINT_ADRESSE      EQU  000A0002H ;Breakpoint-Adresse im DR1-Register
000B0002             27      DR2_BREAKPOINT_Adresse      EQU 000B0002H  ;Breakpoint-Adresse im DR2-Register
000C0000             28      DR3_BREAKPOINT_ADRESSE      EQU 000C0000H  ;Breakpoint-Adresse im DR3-Register
                     29
                     30      ASSUME DS:DATEN
                     31
.......              32      DATEN SEGMENT RW USE32
                     33
00000000 55000000    34      BREAKPOINT_BEDINGUNG_0 DR7_FELDER <0,0,    0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1>
                     35                                                 ;L3,L2,L1,L0=1
                     36
                     37                                                 ;Die Breakpoint-Adressen in DR0..DR7
                     38                                                 ;beziehen sich wegen RW0..RW1=100 und
                     39                                                 ;LEN0..LEN3=00 nur auf Befehle
                     40                                                 ;Der Zugriff auf die Debug-Register
                     41                                                 ;ist wegen GD=0 erlaubt
                     42                                                 ;GE(X)=0 und LE(X)=0 sind bei Befehls-
                     43                                                 ;Breakpoints ohne Bedeutung
                     44                                                 ;Wegen G0..G3=0 und L0..L1=3 gelten
                     45                                                 ;die Breakpoints nur in der augen-
                     46                                                 ;blicklichen Task
.......              47      DATEN ENDS
                     48
.......              49      BREAKPOINT SEGMENT ER USE32
                     50
```

```
00000000  B801000A00        51        MOV EAX,DR0_BREAKPOINT_ADRESSE    ;DR0 = Breakpoint-Adresse 0
00000005  0F23C0           52        MOV DR0,EAX
00000008  B802000A00       53        MOV EAX,DR1_BREAKPOINT_ADRESSE    ;DR1 = Breakpoint-Adresse 1
0000000D  0F23C8           54        MOV DR1,EAX
00000010  B80200800        55        MOV EAX,DR2_BREAKPOINT_ADRESSE    ;DR2 = Breakpoint-Adresse 2
00000015  0F23D0           56        MOV DR2,EAX
00000018  B800000C 00      57        MOV EAX,DR3 _BREAKPOINT_Adresse   ;DR3 = Breakpoint-Adresse 3
0000001D  0F23D8           58        MOV DR3,EAX
00000020  A100000000   R   59        MOV EAX,BREAKPOINT_BEDINGUNG_0    ;DR7 = 1te Breakpoint-Bedingung
00000025  0F23F8   60      MOV DR7,EAX
                           61
.....                      62        BREAKPOINT ENDS
***       WARNING #377 IN 62, (PASS 2) SEGMENT CONTAINS PRIVILEGED INSTRUCTION(S)
                           63
                           64                  END
ASSEMBLY COMPLETE,         1 WARNING,          NO ERRORS.
```

Wie zu erkennen ist, benützt das Programm die ASM386/486-RECORD-Direktive. Sie hat den Vorteil, daß die individuellen Breakpoint-Bedingungs-Felder im Debug-Steuer-Register DR7 bequem erreicht und verändert werden können. Dies geschieht über die RECORD-Initialisierungs-Zuweisung < [Wert] [,.....] >, mit der sich beliebig viele Record-Variablen

(BREAKPOINT_BEDINGUNG_0 ... BREAKPOINT_BEDINGUNG_N)

definieren lassen.

Ich danke Ihnen für Ihr Interesse und Ihre Geduld!

Klaus-Dieter Thies