Application Note AN083
Assembler-Programmierung auf dem MAX-PC

Autor: HB (< HK) Datei: AN083_C.doc (50 Seiten)

1. Assembler-Programmierung

1.1. System-Subroutinen

Auf fast alle Strukturen des Betriebssystems und auf die Strukturen jeder Task, also deren Datenbereich, Parameterbereich und Prozeduren bzw. Funktionen kann von jeder Task aus über die folgenden Subroutinen zugegriffen werden (einschließlich PDT und TDT).

Die Subroutinen sind aus Geschwindigkeitsgründen in Assembler geschrieben, ebenso wie das gesamte Betriebssystem. Aus dem gleichen Grund wurden andere Aufrufkonventionen gewählt als Sie sie vielleicht von MS-DOS her kennen.

Hin- und Rückgabeparameter werden in Registern oder auch in einigen Fällen auf dem Stack übergeben. Gegebenenfalls wird ein Pointer übergeben, dann meist in eax. Adressen werden (bis auf wenige Ausnahmen) als physikalische 32-Bit-Adressen angegeben. Auch die internen Pointer des Betriebssystems werden als physikalische 32-Bit-Adressen gehalten. Die max. Blockgröße für Blocklese- und Blockschreibroutinen beträgt 64 kByte - 256 Byte = 65280 Byte, z.B. bei READ_DATA_BLOCK. Es widerspricht der Philosophie des Betriebssystems, sehr große Blökke "am Stück" zu übertragen. Deshalb wird auch hier empfohlen, nur kleinere Blöcke zu übertragen oder die entsprechende Routine mehrmals aufzurufen. Aus Geschwindigkeitsgründen wird in den Routinen auf die Überprüfung der Blockgröße verzichtet, es wird also kein Fehler gemeldet.

Flags (f): Nach Rückkehr aus einer Subroutine zeigt das Carry-Flag (CY) immer an, ob die Routine ohne Fehler ausgeführt werden konnte. Wenn CY = 1 gesetzt ist, ist ein Fehler aufgetreten. In Register ax steht dann eine Erklärung zum Fehler. Die Fehlercodes entsprechen denen, die auch bei Makrobefehlen auftreten können (siehe Anhang B des MAX6pci-Handbuchs).

Alle Subroutinen lassen den Status des maskierbaren CPU-Interrupts, also das Interrupt Enable Flag (IF), und das Direction Flag (DF) unverändert. Einige Subroutinen verändern diese Flags vorübergehend, hinterlassen sie aber so, wie sie sie vorgefunden haben. Keine der Subroutinen ändert den Status des Nicht Maskierbaren Interrupts (NMI).

Alle Aufrufe von Subroutinen geschehen indirekt über Pointer, die ab 00:800h am Anfang des Speicherbereichs stehen. Jeder Pointer besteht aus 4 Byte (Segment:Offset).

Bei den Subroutinen mit Parameterübergabe in Registern sieht der Aufruf aus einem Assembler-Programm, z.B. zum Setzen eines Parameterbyte einer beliebigen Task, folgendermaßen aus (Borland Turbo Assembler, Ideal Modus):

```
push   ds                          ; ein Segmentregister,
xor    ax,ax                       ;  z.B. ds, retten und
mov    ds,ax                       ;  = 0 setzen

mov    dx,TASK_NR                  ; Ziel-Task
mov    bx,PARAMETER_NR             ; rel. Adresse
mov    al,DATA                     ; neuer Wert (Byte)
call   [DWORD FAR ds: 800h + SUBROUTINE_NR * 4]
                                   ; = db 3eh,0ffh,1eh,16*4,8
pop    ds
jc     FEHLER                      ; Fehler?
....   .......                     ; nein
```

In Kaptitel 1.2 finden Sie ein komplettes Beispielprogramm in Assembler.

Folgende Routinen sind vorhanden (Betriebssystemversion MAX-11A.01x, x=E: ROM-Version, x=R: Download-Version, x=B: Beta-Version):

Nr.	Adr.	Name	Funktion (Kurzbeschreibung)	Seite
0	800h	FORCE_ERROR	Provoziert eine Fehlermeldung	6
1	804h	PROG_IN_ROM	Melde, ob Programm im ROM ist	6
2	808h	INSTALL_TASK	Installiere Programm unter Task	6
3	80ch	GET_TASK_INFO	Info über eine Task (z.B. TDT, PDT)	8
4	810h	GET_TASK_STATUS	Melde, ob Task aktiviert ist	8
5	814h	WAKEUP_TASK	Aktiviere Task	8
6	818h	SLEEP_TASK	Deaktiviere Task	9
7	81ch	-	Reserviert	
8	820h	-	Reserviert	
9	824h	CALL_PROC	Prozedur einer Task aufrufen	10
10	828h	CALL_FUNC	Funktion einer Task aufrufen	10
11	82ch	GET_PAR_ADDRESS	Melde Adresse eines Parameters	11
12	830h	READ_PAR_BYTE	Lies Parameterbyte	12
13	834h	READ_PAR_WORD	Lies Parameterwort	12
14	838h	READ_PAR_DWORD	Lies Parameterdoppelwort	12
15	83ch	READ_PAR_BLOCK	Lies Parameterblock	13
16	840h	WRITE_PAR_BYTE	Schreibe Parameterbyte	13
17	844h	WRITE_PAR_WORD	Schreibe Parameterwort	13
18	848h	WRITE_PAR_DWORD	Schreibe Parameterdoppelwort	13
19	84ch	WRITE_PAR_BLOCK	Schreibe Parameterblock	14
20	850h	-	Reserviert	
21	854h	-	Reserviert	
22	858h	-	Reserviert	
23	85ch	-	Reserviert	
24	860h	READ_DATA_BYTE	Lies Datenbyte	14
25	864h	READ_DATA_WORD	Lies Datenwort	15
26	868h	READ_DATA_DWORD	Lies Datendoppelwort	15
27	86ch	READ_DATA_BLOCK	Lies Datenblock	15
28	870h	WRITE_DATA_BYTE	Schreibe Datenbyte	15
29	874h	WRITE_DATA_WORD	Schreibe Datenwort	16
30	878h	WRITE_DATA_DWORD	Schreibe Datendoppelwort	16
31	87ch	WRITE_DATA_BLOCK	Schreibe Datenblock	16
32	880h	ALLOCATE_RAM	Reserviere freien Speicher	17
33	884h	MASK_INT	Maskiere einen Interrupt	18
34	888h	UNMASK_INT	Demaskiere einen Interrupt	18
35	88ch	CLEAR_INT	Lösche einen Pending Interrupt	19
36	890h	END_OF_INT	Beende Interrupt-Service-Routine	19
37	894h	-	Reserviert	
38	898h	SET_INT_MODE	Interrupt Priorität setzen (**Noch nicht implementiert)**	
39	89ch	TRIGGER_WATCHDOG	Watchdog nachtriggern	19
40	8a0h	CACHE_CONTROL	Cache steuern	20
41	8a4h	LOCAL_LED_ON	LED (auf MAX-PC) "ein"	20
42	8a8h	LOCAL_LED_OFF	LED (auf MAX-PC) "aus	20
43	8ach	-	Reserviert	

Nr.	Adr.	Name	Funktion (Kurzbeschreibung)	Seite
44	8b0h	-	Reserviert	
45	8b4h	GET_RTC_STATUS	Lies Status der Uhr	20
46	8b8h	SET_RTC_MODE	Setze Mode von Uhr	21
47	8bch	GET_DATE_AND_TIME	Lies Datum und Uhrzeit	21
48	8c0h	SET_DATE_AND_TIME	Setze Datum und Uhrzeit	22
49	8c4h	GET_TIMER	Lies Timer (Zähler/Status)	23
50	8c8h	SET_TIMER	Setze Timer (Mode/Anfang)	23
51	8cch	READ_EEPROM_BLOCK	Lies Block aus EEPROM	24
52	8d0h	WRITE_EEPROM_BLOCK	Schreibe Block in EEPROM	24
53	8d4h	SEND_HOST_SRQ	SRQ an Host schicken	26
54	8d8h	GET_TDT_ADDRESS	Lies Adresse einer TDT	25
55	8dch	-	Reserviert	
56	8e0h	-	Reserviert	
57	8e4h	-	Reserviert	
58	8e8h	GET_INT_TASK	Melde Task, die Interrupt nutzt	25
59	8ech	CONVERT_TIMER_DATA	Berechne Timer-Wert	25
60	8f0h	SEND_BUFFER_SRQ	Sende einen gepufferten SRQ	26
61	8f4h	MACRO_LOCKING	Makro-Befehle unterbrechbar/nicht	27
65	904h	WAKEUP_TI_TASK	TI-Task aktivieren	27
66	908h	SLEEP_TI_TASK	TI-Task deaktivieren (**Noch nicht implementiert)**	
70	918h	CREATE_BUFFER	Ringpuffer erzeugen	28
71	91ch	DELETE_BUFFER	Ringpuffer entfernen	29
72	920h	CLEAR_BUFFER	Ringpuffer komplett leeren	29
73	924h	GET_BUFFER_STATUS	Anzahl Zeichen im Ringpuffer bestimmen	30
74	928h	WRITE_BUFFER_BYTE	Ein Byte in Ringpuffer schreiben	30
75	92ch	WRITE_BUFFER_WORD	Ein Wort in Ringpuffer schreiben	30
76	930h	WRITE_BUFFER_DWORD	Ein Doppelwort in Ringpuffer schreiben	30
77	934h	WRITE_BUFFER_BLOCK	Block in Ringpuffer schreiben	31
78	938h	WRITE_BUFFER_MAX	Block in Ringpuffer schreiben (max.)	31
79	93ch	READ_BUFFER_BYTE	Ein Byte aus Ringpuffer lesen	32
80	940h	READ_BUFFER_WORD	Ein Wort aus Ringpuffer lesen	32
81	944h	READ_BUFFER_DWORD	Ein Doppelwort aus Ringpuffer lesen	32
82	948h	READ_BUFFER_BLOCK	Block aus Ringpuffer lesen	32
83	94ch	READ_BUFFER_MAX	Block aus Ringpuffer lesen (max.)	33
84	950h	VIEW_BUFFER_BLOCK	Block aus Ringpuffer kopieren	33
85	954h	VIEW_BUFFER_MAX	Block aus Ringpuffer kopieren (max.)	34
90	968h	GET_TASK_NUMBER	Tasknummer zu Programmnummer ermitteln	34
91	96ch	GET_INT_VECTOR	Interrupt-Vektor lesen	37
92	970h	SET_INT_VECTOR	Interrupt-Vektor setzen	37
93	974h	INIT_IO	Initialisieren eines Moduls	36
96	980h	INSTALL_HOST	Host installieren	37
97	984h	UNINSTALL_HOST	Host deinstallieren	38
98	988h	GET_HOST_INFO	Info von Host lesen	38
99	98ch	CALL_CMD	Makrobefehl ausführen	39
103	99ch	MOVE_MEMORY_BLOCK	Speicher kopieren	39
104	9a0h	FLASH_STATUS	Status von Flash lesen	40

Nr.	Adr.	Name	Funktion (Kurzbeschreibung)	Seite
105	9a4h	FLASH_ERASE	Block in Flash löschen	40
106	9a8h	FLASH_PROGRAM	Block in Flash programmieren	40
107	9ach	FLASH_READ	Block aus Flash lesen	41
112	9c0h	GET_PAR_SEMA	Semaphore für Parameterbereich einer Task holen	11
113	9c4h	RESET_PAR_SEMA	Semaphore für Parameterbereich einer Task freigeben	12
114	9c8h	GET_DATA_SEMA	Semaphore für Datenbereich einer Task holen	14
115	9cch	RESET_DATA_SEMA	Semaphore für Datenbereich einer Task freigeben	14
116	9d0h	-	Reserviert	
117	9d4h	-	Reserviert	
118	9d8h	GET_RTC_INT_STATUS-	Interrupt-Status des RTC melden	20
119	9dch	GET_TASKS_INSTALLED	Liefere installierte Tasks	41
120	9e0h	READ_CMOS_RAM	CMOS-RAM lesen	23
121	9e4h	WRITE_CMOS_RAM	CMOS-RAM schreiben	23
122	9e5h	SEND_MESSAGE	Message senden?	
124	9f0h	READ_INDEX_PORT	Lies von Portadresse indiziert	41
125	9f4h	WRITE_INDEX_PORT	Schreibe auf Portadresse indiziert	42
126	9f8h	READ_KONFIG	Konfigurationsregister lesen	42
127	9fch	WRITE_KONFIG	Konfigurationsregister schreiben	42
129	a04h	-	Reserviert	
130	a08h	SET_MODULE_AD	**Noch nicht implementiert**	
131	a0ch	GET_ALARM	Lies Alarmzeit aus RTC	22
132	a10h	SET_ALARM	Setze Alarmzeit in RTC	22

FORCE_ERROR (Nr. 0)

Diese Subroutine dient Testzwecken und liefert nur eine Fehlermeldung zurück. Bei der Rückgabe ist CY immer = 1.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	CY=0 kommt nicht vor
Exit (CY=1):	ax	Fehlercode: 0fe0h = Subroutine nicht implementiert

PROG_IN_ROM (Nr. 1)

Diese Subroutine prüft, ob ein bestimmtes Programm im ROM enthalten ist.

Entry:	ax	Programm-Nr.
Changed:	f, ax	
Exit (CY=0):	ax	0: Programm nicht vorhanden unverändert: Programm ist im ROM
Exit (CY=1):	ax	Fehlercode: 0fe0h = Subroutine nicht implementiert

INSTALL_TASK (Nr. 2)

INSTALL_TASK installiert ein Programm unter einer Task. Die Installierung geschieht entsprechend einem Record, auf den das Register eax zeigt (eax = 32 Bit physikal. Adresse). Es sind die gleichen Installierungen und Optionen möglich wie mit dem Makrobefehl 40h. INSTALL_TASK darf nicht in einer Auto-Init-Routine aufgerufen werden.

Entry:	eax	Physikal. Adresse (32 Bit) des Records
Changed:	f, eax	
Exit (CY=0):	ax	Programm installiert, ax = Task-Nr.
Exit (CY=1):	ax	Fehlercode (ein Fehler kann auch von Auto-Init oder PREPARE kommen)

Der Aufbau des Installierungsrecords in Kurzform. Der Record hat eine Länge von 22 Byte. Die Bedeutung der Adresse *a* in diesem Record richtet sich nach der Art der zu installierenden Datei, was in den Flags *f* vermerkt ist.

Offset	**Typ**	**Bedeutung**
0	Byte	Länge des Records, z.Zt. = 22
1	Byte	Typ der TDT, z.Zt. = 1
2	Byte	Länge der TDT, z.Zt. = 36
3	Byte	Reserviert = 0
4	Wort	Tasknummer (16 bis 1023, Task 0 bis 20 reserviert)
6	Wort	Programm-Nummer (1 bis 65534, siehe auch PDT)
8	Byte	Interrupt-Nummer (nur bei DI- bzw. II-Tasks)
9	Byte	Reserviert = 0
10	Doppelwort	Flags *f*, Erklärungen siehe Anhang D des MAX6pci-Handbuchs Bit 11: 1 = Auto-Init aufrufen Bit 12: 1 = Task sofort aktivieren
14	Doppelwort	Größe des Datenbereichs. Die Größe wird in Anzahl Byte angegeben. Weitere Erklärungen finden sich bei Flags zu Bit 9 und 10 (s.o.).
18	Doppelwort	Adresse *a*, Format und Wert abhängig von Flags. Die Adresse wird als physikalische oder Segment:Offset-Adresse angegeben (siehe Flags, Bit 6 bis 8, nächste Tabelle). Bei Verwendung der mitgelieferten Bibliothek wird die Formatanpassung automatisch übernommen.

Wo?	**Programmformat**	**Adresse a**	**Format Adresse**	**Flag-Bit* 8 7 6**	**Typ**
RAM	PDT tiny	Anfang PDT	physikal.	0 0 0	Assembler
RAM	PDT large	Anfang PDT	physikal.	0 0 0	Assembler
RAM	EXE not relocated	Anfang EXE-Header	physikal.	1 1 0	C / Pascal
RAM	EXE relocated	START_UP Code	Seg:Offs.	0 1 0	Reserviert
ROM	PDT	wird ignoriert	-	0 0 1	Reserviert

* alle anderen Kombinationen sind zur Zeit nicht erlaubt

GET_TASK_INFO (Nr. 3)

GET_TASK_INFO liefert Informationen über eine Task, z.B. über die Task-Deskriptor-Tabelle (TDT) oder die Programm-Deskriptor-Tabelle (PDT). Wenn kein Programm unter der Task installiert ist, erfolgt eine Fehlermeldung. Die angefragte Task kann auch deaktiviert sein.

Entry:	dx	Tasknummer
	ax	Bestimmt die Art der Information (siehe Anhang C des MAX6pci-Handbuchs)
	ah = 0:	Inhalt der PDT lesen: al = rel. Adresse des PDT-Eintrags
	ah = 1:	Inhalt der TDT lesen: al = rel. Adresse des TDT-Eintrags
	ah = 2:	Adresse der TDT: al = 0
	ah = 3:	Inhalt der DDT (Debug-Descriptor-Table): al = rel. Adresse des DDT-Eintrags
Changed:	f, eax, edx	
Exit (CY=0):	eax	4 Byte Info (siehe Anhang C des MAX6pci-Handbuchs)
Exit (CY=1):	ax	Fehlercode: 08e0h = kein Programm installiert 19e0h = keine DDT angelegt

GET_TASK_STATUS (Nr. 4)

GET_TASK_STATUS meldet die Anzahl der Aktivierungen einer Task. Interrupt-Tasks können nur einmal aktiviert werden, NI-Tasks auch mehrfach.

Entry:	dx	Tasknummer
Changed:	f, ax, edx	
Exit (CY=0):	ax	Anzahl der Aktivierungen
Exit (CY=1):	ax	Fehlercode: 08d2h = Task nicht installiert

WAKEUP_TASK (Nr. 5)

WAKEUP_TASK aktiviert eine NI- oder II-Task einmal. NI-Tasks können auch mehrfach aktiviert werden (max. 255). Sie werden dann vom Task-Scheduler des Betriebssystems entsprechend oft aufgerufen. TI-Tasks können mit dieser Subroutine nicht aktiviert werden (siehe hierzu WAKEUP_TI_TASK (Nr. 65)).

Eine NI-Task kann mit dieser Subroutine auch einmalig in der Bearbeitung durch den Scheduler vorgezogen werden (ax = 2 oder ax = 3). Die Task kommt dann automatisch als nächste NI-Task an die Reihe, wenn die aktuell laufende NI-Task beendet wurde. Mit "Prüfung" (ax = 3) wird, wenn bereits eine NI-Task vorgezogen, aber noch nicht ausgeführt wurde, eine Fehlermeldung zurückgeliefert (ax = 20d4h). Falls die vorgezogene NI-Task nicht aktiviert war, wird sie nur ein einziges mal aufgerufen und dann wieder deaktiviert.

Entry:	dx	Tasknummer
	ax = 1	Aktivieren der Task (NI- oder II-Task)
	ax = 2	Einmaliges Vorziehen einer NI-Task (in jedem Fall)

	ax = 3	Einmaliges Vorziehen einer NI-Task mit Prüfung, ob bereits eine NI-Task auf vorgezogene Abarbeitung wartet. Wenn ja, erfolgt eine Fehlermeldung.	
Changed:	f, ax		
Exit (CY=0):	-		
Exit (CY=1):	ax	Fehlercode:	10e0h = Falscher Parameter 20d4h = Warnung: schon benutzt 09d2h = Nicht implementiert 08d2h = Task nicht installiert 0ad2h = NI-Task-Tabelle voll

SLEEP_TASK (Nr. 6)

SLEEP_TASK deaktiviert eine Task einmal. Wenn eine NI-Task mehrfach aktiviert wurde, muß sie auch entsprechend oft wieder deaktiviert werden, damit sie inaktiv ist.

Entry:	dx	Tasknummer	
	ax	Immer = 1 (dies ist für zukünftige Entwicklungen vorgesehen)	
Changed:	f, ax		
Exit (CY=0):	-		
Exit (CY=1):	ax	Fehlercode:	0bd2h = Systemfehler 08d2h = Task nicht installiert 0cd2h = Task nicht aktiv

CALL_PROC (Nr. 9)

CALL_PROC ruft eine Prozedur einer Task auf. Es werden keine Parameter an die Prozedur übergeben und keine zurückerwartet, außer ggfls. eine Fehlermeldung.

Entry:	dx	Tasknummer
	cx	Prozedur-Nr. (0, 1, 2, ...)
Changed:	f, eax, bx, cx (wenn die Prozedur nur retf macht).	
Exit (CY=0):	-	Kein Fehler aufgetreten
Exit (CY=1):	ax	Fehlercode: 18e0h = "Prozedur bzw. Funktion nicht vorhanden" (tritt auch auf, wenn gar kein Programm unter der Task installiert ist). In diesem Fall wird die Prozedur nicht mehr aufgerufen.

Zu Beginn der aufgerufenen Prozedur enthält ax die Tasknummer und cx die Prozedurnummer. Alle in der aufgerufenen Prozedur verwendeten Register müssen am Anfang gerettet und am Ende wieder restauriert werden (nicht nötig für f, eax, bx, cx, dx, ds).

CALL_FUNC (Nr. 10)

CALL_FUNC ruft eine Funktion einer Task auf, gegebenenfalls mit Parameterübergabe an die Funktion und/oder auch mit Parameterrückgabe zum aufrufenden Programm. Das aufrufende Programm muß jeweils einen Puffer für die Parameter 'Hin' und 'Zurück' zur Verfügung stellen, wenn Parameter übergeben bzw. zurückerwartet werden.

Die Überprüfung, ob die aufgerufene Funktion auch die Erwartungen des aufrufenden Programmes bzgl. Übergabeparameter erfüllen kann, wird von der aufgerufenen Funktion selbst vorgenommen (anders als bei C mit Prototyp, dort übernimmt das der Compiler). Zu Beginn der aufgerufenen Funktion sind dx, di, si, eax und ecx so gesetzt, wie sie an CALL_FUNC übergeben werden. In der aufgerufenen Funktion müssen die verwendeten Register am Anfang gerettet und am Ende wieder restauriert werden (nicht nötig für f, eax, ecx, bx, di, si, ds). Alle Register (außer f, sp und ds) werden so zum Aufrufenden zurückgegeben, wie sie die aufgerufene Funktion am Ende hinterlassen hat. Die aufgerufene Funktion kann einen Fehler an den Aufrufenden zurückliefern. Hierzu muß sie am Ende CY = 1 und ax = Fehlermeldung setzen. Die aufgerufene Funktion kann im Fehlerfall keine Parameter 'Zurück' zum Aufrufenden zurückliefern.

Funktionen können auch vom PC aus per Makrobefehl aufgerufen werden. Wenn die Funktion einen Fehler zum PC melden will, kann sie dies nur durch eine 1-Byte-Fehlermeldung tun. Die von der aufgerufenen Funktion gelieferte Fehlermeldung muß vom Typ xxe0h oder xxdeh sein, dann wird das Byte xxh zum PC geliefert. Bei allen anderen Fehlergruppen wird xxh = 1ah "unbekannter Fehler" zum PC geliefert.

Entry:	dx	Tasknummer
	bx	Funktionsnummer (2, 3, 4, ...)
	di	Anzahl Byte, die der Funktion zur Verfügung gestellt werden (Parameter 'Hin')
	si	Anzahl Byte, die die Funktion zurückliefern soll (Parameter 'Zurück')
	eax	Zeiger auf den Puffer für die Parameter 'Hin' (physikalische Adresse)
	ecx	Zeiger auf den Puffer für die Parameter 'Zurück' (physikalische Adresse)

Changed:		Alle Register (dx, bx, di, si, eax und ecx) werden so zum Aufrufenden zurückgegeben, wie sie die aufgerufene Funktion am Ende hinterlassen hat. Wenn die aufgerufene Funktion nur CY = 0, si = 0 setzt und dann retf macht, sind nur f, bx und ggfls. si geändert. Wenn die aufgerufene Funktion zum Melden eines Fehlers CY = 1, ax = xxe0h oder xxdeh setzt und dann retf macht, sind nur f, bx und ax geändert.
Exit (CY=0):	di	Rest (Anzahl Byte) der von der aufgerufenen Funktion nicht verwendeten Byte 'Hin' (die Parameter am Anfang des Puffers werden von der aufgerufenen Funktion zuerst verwendet)
	si	Anzahl der von der aufgerufenen Funktion tatsächlich zurückgelieferten Byte 'Zurück'
Exit (CY=1):	ax	Fehler aufgetreten, ax = Fehlercode Der Fehler kann entweder beim Aufruf der Funktion auftreten (z.B. Fehlercode = 18e0h: "Prozedur bzw. Funktion nicht vorhanden oder kein Programm unter der Task installiert"), oder die Funktion kann durch CY = 1 und ax = xxe0h oder xxdeh (xx = Fehlertyp bzw. Parameter-Offset) einen Fehler zurückliefern. Der Aufrufende muß die Auswertung der Fehlermeldung selbst übernehmen.

GET_PAR_ADDRESS (Nr. 11)

GET_PAR_ADDRESS ermittelt die physikalische 32-Bit-Adresse eines Parameters einer Task. Wenn eax = 0 gesetzt wird, wird die Adresse des Anfangs des Parameterbereichs gemeldet.

Entry:	edx	Tasknummer
	eax	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
Changed:	f, eax	
Exit (CY=0):	eax	Adresse des Parameters (physikal.)
Exit (CY=1):	ax	Fehlercode

GET_PAR_SEMA (Nr. 112)

Diese Funktion versucht, die Semaphore für den Parameterbereich der Task t zu holen.

Entry:	ax	Task-Nr. t des Parameterbereichs
Changed:	f, eax	
Exit (CY=0):	ax	Ergebnis: 0 = ok, Semaphore bekommen <> 0 = Semaphore nicht bekommen
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden

RESET_PAR_SEMA (Nr. 113)

Diese Funktion gibt die Semaphore für den Parameterbereich der Task t wieder frei.

Entry:	ax	Task-Nr. t des Parameterbereichs
Changed:	f, eax	
Exit (CY=0):	-	kein Fehler
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden

READ_PAR_BYTE (Nr. 12)

READ_PAR_BYTE liest den Wert eines Parameterbyte einer Task.

Entry:	edx	Tasknummer
	eax	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
Changed:	f, eax	
Exit (CY=0):	al	Data
Exit (CY=1):	ax	Fehlercode

READ_PAR_WORD (Nr. 13)

READ_PAR_WORD liest den Wert eines Parameterwortes einer Task.

Entry:	edx	Tasknummer
	eax	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
Changed:	f, eax	
Exit (CY=0):	ax	Data
Exit (CY=1):	ax	Fehlercode

READ_PAR_DWORD (Nr. 14)

READ_PAR_DWORD liest den Wert eines Parameterdoppelwortes (4 Byte) einer Task.

Entry:	edx	Tasknummer
	eax	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
Changed:	f, eax	
Exit (CY=0):	eax	Data
Exit (CY=1):	ax	Fehlercode

READ_PAR_BLOCK (Nr. 15)

READ_PAR_BLOCK kopiert einen Datenblock aus dem Parameterbereich einer Task an die durch den Pointer in eax gegebene physikalische Adresse. Der Parameterbereich ist während des Zugriffs nicht vor dem Zugriff durch eine andere Task geschützt. Das könnte z.B. durch Maskieren des CPU-Interrupts vor dem Aufruf geschehen, oder durch Setzen der Semaphore.

Entry:	edx	Tasknummer (Quelle)
	ebx	Parameternummer des ersten zu kopierenden Parameters (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
	cx	Anzahl zu kopierender Parameterbyte (max. 64K)
	eax	Zieladresse (physikalisch)
Changed:	f, eax,ebx, cx, edx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_PAR_BYTE (Nr. 16)

WRITE_PAR_BYTE schreibt ein Parameterbyte.

Entry:	edx	Tasknummer
	ebx	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
	al	Data
Changed:	f, ebx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_PAR_WORD (Nr. 17)

WRITE_PAR_WORD schreibt ein Parameterwort.

Entry:	edx	Tasknummer
	ebx	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
	ax	Data
Changed:	f, ebx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_PAR_DWORD (Nr. 18)

WRITE_PAR_DWORD schreibt ein Parameterdoppelwort (4 Byte).

Entry:	edx	Tasknummer
	ebx	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
	eax	Data
Changed:	f, ebx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_PAR_BLOCK (Nr. 19)

WRITE_PAR_BLOCK kopiert einen Datenblock ab der durch den Pointer in eax angegebenen physikalischen Adresse in den Parameterbereich einer Task. Der Parameterbereich ist während des Zugriffs nicht vor dem Zugriff durch eine andere Task geschützt. Das könnte z.B. durch Maskieren des CPU-Interrupts vor dem Aufruf geschehen, oder durch Setzen der Semaphore.

Entry:	edx	Ziel-Tasknummer
	ebx	Parameternummer des ersten zu schreibenden Parameters (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
	cx	Anzahl zu kopierender Byte (max. 64K)
	eax	Quelladresse (physikalisch)
Changed:	f, eax, ebx, cx, edx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

GET_DATA_SEMA (Nr. 114)

Diese Funktion versucht, die Semaphore für den Datenbereich der Task t zu holen.

Entry:	ax	Task-Nr. t des Datenbereichs
Changed:	f, eax	
Exit (CY=0):	ax	Ergebnis: 0 = ok, Semaphore bekommen <> 0 = Semaphore nicht bekommen
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden

RESET_DATA_SEMA (Nr. 115)

Diese Funktion gibt die Semaphore für den Datenbereich der Task t wieder frei.

Entry:	ax	Task-Nr. t des Datenbereichs
Changed:	f, eax	
Exit (CY=0):	-	kein Fehler
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden

READ_DATA_BYTE (Nr. 24)

READ_DATA_BYTE liest den Wert eines Datenbyte einer Task von der durch den Anfang Datenbereich + Offset gegebenen Adresse.

Entry:	edx	Tasknummer
	eax	Offset
Changed:	f, eax, edx	
Exit (CY=0):	al	Data
Exit (CY=1):	ax	Fehlercode

READ_DATA_WORD (Nr. 25)

READ_DATA_WORD liest den Wert eines Datenwortes einer Task von der durch den Anfang Datenbereich + Offset gegebenen Adresse.

Entry:	edx	Tasknummer
	eax	Offset
Changed:	f, eax, edx	
Exit (CY=0):	ax	Data
Exit (CY=1):	ax	Fehlercode

READ_DATA_DWORD (Nr. 26)

READ_DATA_DWORD liest den Wert eines Datendoppelwortes einer Task von der durch den Anfang Datenbereich + Offset gegebenen Adresse.

Entry:	edx	Tasknummer
	eax	Offset
Changed:	f, eax, edx	
Exit (CY=0):	eax	Data
Exit (CY=1):	ax	Fehlercode

READ_DATA_BLOCK (Nr. 27)

READ_DATA_BLOCK kopiert einen Datenblock aus dem Datenbereich einer Task von der durch den Anfang Datenbereich + Offset dieser Task gegebenen Adresse an die durch den Pointer in eax gegebene physikalische Adresse. Der Datenbereich ist während des Zugriffs nicht vor dem Zugriff durch andere Tasks geschützt. Das könnte z.B. durch Maskieren des CPU-Interrupts vor dem Aufruf geschehen, oder durch Setzen der Semaphore.

Entry:	edx	Tasknummer (Quelle)
	ebx	Offset
	ecx	Anzahl zu kopierender Byte
	eax	Zieladresse (physikalisch)
Changed:	f, eax, ebx, ecx, edx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_DATA_BYTE (Nr. 28)

WRITE_DATA_BYTE schreibt ein Byte in den Datenbereich einer Task an die durch den Anfang Datenbereich + Offset gegebene Adresse.

Entry:	edx	Tasknummer
	ebx	Offset
	al	Data
Changed:	f, ax, edx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_DATA_WORD (Nr. 29)

WRITE_DATA_WORD schreibt ein Wort in den Datenbereich einer Task an die durch den Anfang Datenbereich + Offset gegebene Adresse.

Entry:	edx	Tasknummer
	ebx	Offset
	ax	Data
Changed:	f, ax, edx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_DATA_DWORD (Nr. 30)

WRITE_DATA_DWORD schreibt ein Doppelwort (4 Byte) in den Datenbereich einer Task an die durch den Anfang Datenbereich + Offset gegebene Adresse.

Entry:	edx	Tasknummer
	ebx	Offset
	eax	Data
Changed:	f, ax, edx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_DATA_BLOCK (Nr. 31)

WRITE_DATA_BLOCK kopiert einen Block von Daten von der durch den Pointer in eax gegebenen physikalische Adresse in den Datenbereich einer Task an die durch den Anfang Datenbereich + Offset dieser Task gegebenen Adresse. Der Datenbereich ist während des Zugriffs nicht vor dem Zugriff durch andere Tasks geschützt. Das könnte z.B. durch Maskieren des CPU-Interrupts vor dem Aufruf geschehen, oder durch Setzen der Semaphore.

Entry:	edx	Tasknummer (Ziel)
	ebx	Offset
	ecx	Anzahl zu kopierender Byte
	eax	Quelladresse (physikalisch)
Changed:	f, eax, ebx, ecx, edx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

ALLOCATE_RAM (Nr. 32)

ALLOCATE_RAM reserviert Speicher auf der Karte. Die Routine kann auch verwendet werden, um nur die Größe des freien RAM zu ermitteln, ohne Speicher zu reservieren.

Für das Reservieren sind verschiedene Strategien möglich:

0 = Größe des freien RAM ermitteln

1 = UP absolut:
Es soll soviel Platz wie angefordert von der unteren Grenze des freien RAM an aufwärts reserviert werden. Wenn nicht genug Platz ist, wird nichts reserviert.

2 = UP max.:
Es soll soviel Platz wie möglich aber nicht mehr als angegeben von der unteren Grenze des freien RAM an aufwärts reserviert werden.

3 = DOWN absolut:
Es soll soviel Platz wie angefordert von der oberen Grenze des freien RAM an abwärts reserviert werden. Wenn nicht genug Platz ist, wird nichts reserviert.

4 = DOWN max.:
Es soll soviel Platz wie möglich aber nicht mehr als angegeben von der oberen Grenze des freien RAM an abwärts reserviert werden.

Alignment:

Hiermit kann die Anfangsadresse des reservierten Bereichs so gewählt werden, daß sie ohne Rest durch n teilbar ist, wobei n = 0, 1, 2, 4, 8, 16, ... sein kann.

Die Angaben von Tasknummer und Art der Verwendung des Speichers dienen betriebssystem-internen Zwecken und können auch weggelassen werden. Sie sind für zukünftige Entwicklungen vorgesehen.

Entry:	eax	Größe bzw. Maximum des zu reservierenden Bereichs (in Anzahl Byte), wird bei bl = 0 ignoriert.
	bl	Strategie: 0: nur Größe freies RAM liefern (Alignment in bh wird berücksichtigt) 1: UP absolut 2: UP max. 3: DOWN absolut 4: DOWN max.
	bh	Byte-Alignment: 2^{bh} (bh max. <= 15) z.B. bh = 2: DWORD (32 Bit)
	cx	Verwendung des Speichers (Statistik)
	dx	Nr. der Task, die den Speicher nutzt (Statistik)
Changed:	f, eax, ebx	
Exit (CY=0):	eax	Größe des freien RAM (bl = 0) bzw. Anfangsadresse des reservierten Bereichs (bl = 1, 2, 3, 4)
	ebx	Unverändert (bl = 0) bzw. Größe des reservierten Bereichs (bl = 1, 2, 3, 4)
Exit (CY=1):	ax	Fehlercode: 13d7h = Nicht genügend Platz 10e0h = falsche Parameter bei Aufruf

MASK_INT (Nr. 33)

Diese Subroutine maskiert einen Interrupt.

Entry:	al	Interrupt-Nr.:		
		2	(= 02h):	NMI
		120	(= 78h):	Timer-A
		121	(= 79h):	XT-Keyboard
		123	(= 7bh):	COM2
		124	(= 7ch):	X-Bus / COM2 (= PIRQ6)
		125	(= 7dh):	X-Bus / EPP (= PIRQ7)
		126	(= 7eh):	Mailbox RBF (=PIRQ4)
		127	(= 7fh):	EPP
		144	(= 90h):	Uhr (RTC)
		145	(= 91h):	X-Bus /Grafik (= PIRQ2)
		146	(= 92h):	X-Bus (= PIRQ3)
		147	(= 93h):	Timer C (= PIRQ5)
		148	(= 94h):	Mouse
		149	(= 95h):	Ext. Input (= PIRQ0)
		150	(= 96h):	Ext. Input HOST (= PIRQ1)
		151	(= 97h):	PC-Card
		152	(= 98h):	RTC-Interrupt: Update-Ended
		153	(= 99h):	RTC-Interrupt: Alarm
		154	(= 9ah):	RTC-Interrupt: Periodic

Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode: 10e0h = falsche Interrupt-Nr.

UNMASK_INT (Nr. 34)

Diese Subroutine demaskiert einen Interrupt.

Entry:	al	Interrupt-Nr.: siehe bei Subroutine MASK_INT
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode: 10e0h = falsche Interrupt-Nr.

CLEAR_INT (Nr. 35)

Diese Subroutine löscht das dem entsprechenden Interrupt-Eingang zugehörige Statusbit im Interrupt Controller, das anzeigt, daß ein Interrupt aufgetreten ist und auf Bedienung wartet.
Zur Erklärung: Die Information, daß ein Interrupt auslösendes Ereignis (also eine aktive Flanke) aufgetreten ist, wird im Interrupt Controller gespeichert, unabhängig davon, ob der Interrupt maskiert ist oder nicht. Wenn er nicht maskiert ist, wird er wie üblich von der CPU, abhängig von der Priorität und dem Interrupt-Flag in der CPU, bedient. Wenn er maskiert war und nun demaskiert wird, wird dieser Interrupt ebenfalls in jedem Fall noch von der CPU bedient, gleichgültig, wann zuvor die aktive Flanke aufgetreten war. Das ist in vielen Fällen unerwünscht. Meistens sollen Interrupts erst ab einem bestimmten Zeitpunkt registriert werden. Dies läßt sich erreichen, wenn das zugehörige Statusbit im Interrupt Controller vorsichtshalber unmittelbar vor dem Demaskieren gelöscht wird.
Sonderfall NMI: Wenn ein "Nicht Maskierbarer" Interrupt (NMI) auftritt, während er maskiert ist, dann wird diese Information nicht wie bei den normalen Interrupts gespeichert. Nach dem Demaskieren wird also kein NMI ausgelöst, erst wenn das Interrupt auslösende Ereignis nach der Demaskierung wieder auftritt.

Entry:	al	Interrupt-Nr.: siehe bei Subroutine MASK_INT (ausgenommen NMI)
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode: 10e0h = falsche Interrupt-Nr.

END_OF_INT (Nr. 36)

Diese Subroutine beendet eine Interrupt-Service-Routine, löscht also den Interrupt, der gerade 'In-Service' ist (das entspricht dem Senden von EOI an den Interrupt-Controller)

Entry:	al	Interrupt-Nr. (siehe Subroutine MASK_INT) (ausgenommen NMI)
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode, z.B. 10e0h = falsche Interrupt-Nummer

TRIGGER_WATCHDOG (Nr. 39)

Diese Subroutine (re)triggert den Watchdog auf dem MAX-PC Modul.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

CACHE_CONTROL (Nr. 40)

Diese Subroutine schaltet den Cache ein bzw. aus.

Entry:	al	0: Status von Cache liefern 1: Cache ausschalten 2: Cache einschalten (Write-through) 3: Cache einschalten (Write-back)
Changed:	f, eax	
Exit (CY=0):	al	Status von Cache (1, 2, 3, s.o.)
Exit (CY=1):	ax	Fehlercode

LOCAL_LED_ON (Nr. 41)

Diese Subroutine schaltet die on-board LED ein.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

LOCAL_LED_OFF (Nr. 42)

Diese Subroutine schaltet die on-board LED aus.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

GET_RTC_STATUS (Nr. 45)

Diese Subroutine liest den Status der Echtzeituhr. Ein Periodic Interrupt, der Alarm Interrupt und der Update-Ended Interrupt der Uhr sind mit IRQ-8 (= IRQ-0 des Interrupt-Slave-Controllers) verbunden. Mögliche Impulsfrequenzen des Periodic Interrupt sind unten angegeben. Ein weiterer Timer kann durch den Alarm realisiert werden, der als Timer allerdings nur 1/s, 1/min oder 1/h einen Interrupt auslösen kann.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	ax	Status, Erklärung s.u.
Exit (CY=1):	ax	Fehlercode: 20e1h = Device wird von OsX benutzt

Die Bits des Statusregisters:

Bit	Bedeutung
0	Daylight Saving (1 = on)
1	1 = 24 Stunden Betriebsart, 0 = 12 Stunden Betriebsart
2	Data Mode (0 = BCD, 1 = Binary)

3	=0 (reserviert)	
4	Update-Ended Interrupt enabled (= 1)	
5	Alarm Interrupt enabled (= 1)	
6	Periodic Interrupt enabled (= 1)	
7	1 = Uhr gestoppt, 0 = Uhr läuft	
11..8	Frequenz am Impulsausgang:	
	0000b = disabled	1000b = 3,906 ms
	0001b = 3,906 ms	1001b = 7,812 ms
	0010b = 7,812 ms	1010b = 15,625 ms
	0011b = 122,070 µs	1011b = 31,25 ms
	0100b = 244,141 µs	1100b = 62,5 ms
	0101b = 488,281 µs	1101b = 125 ms
	0110b = 976,563 µs	1110b = 250 ms
	0111b = 1,953 ms	1111b = 500 ms
14..12	Betriebsart, Standardmäßig = 010b	
15	Update in Progress (= 1)	

GET_RTC_INT_STATUS (Nr. 118)

Dieses Register kann nur gelesen werden. Alle Bits werden durch das Lesen gelöscht..

Entry:		
Changed:	f, ax	
Exit (CY=0):	al	Bit 7: Interrupt Request Flag (wird nur gesetzt, wenn einer der Interrupts im RTC enabled ist) Bit 6: Periodic Interrupt Flag Bit 5: Alarm Interrupt Flag Bit 4: Update-Ended Flag Bit 3..0: reserviert (= 0)
Exit (CY=1):	ax	Fehlercode, z.Zt. keiner definiert

SET_RTC_MODE (Nr. 46)

Die Betriebsart der Uhr wird entsprechend dem Bitmuster in ax gesetzt.

Entry:	ax	Mode, siehe bei GET_RTC_MODE (Nr. 45). Für Standard-Betrieb sollten Bit 15..12 = 0010b gesetzt werden
Changed:	f, ax	
Exit (CY=0):		
Exit (CY=1):	ax	Fehlercode: 20e1h = Device wird von OsX benutzt

GET_DATE_AND_TIME (Nr. 47)

Diese Subroutine liest das Datum, den Wochentag und die Zeit aus der Uhr. Jedes Byte der Register eax und ebx enthält eine Angabe in binärer Form.

Beispiele: Die Sekunden werden in al geliefert mit einem Wertebereich von 0 bis 59 (00h bis 3bh). Der Wochentag steht in bl mit einem Wertebereich von 0 (Sonntag) bis 6 (Samstag). Das

Jahr steht in Bit 24 bis 31 von Register ebx mit einem Wertebereich von 00 bis 99 (00 bis 63h). Die Stunden werden im 24-Stunden-Mode von 0 bis 23 angegeben, im 12-Stunden-Mode zeigt Bit 7 AM/PM an (0 = AM, 1 = PM), Bit 3..0 die Stunden von 1..12.

Entry:	-		
Changed:	f, eax, ebx		
Exit (CY=0):	eax	Jahrhundert ^ Stunden ^ Minuten ^ Sekunden	
	ebx	Jahr ^ Monat ^ Tag ^ Wochentag	
		(bei Wochentag: 0 = Sonntag, 1 = Montag ...)	
Exit (CY=1):	ax	Fehlercode:	22e1h = Device nicht zugreifbar
			21e1h = unerlaubte Zeitangabe von Device
			20e1h = Device wird von OsX benutzt

SET_DATE_AND_TIME (Nr. 48)

Diese Subroutine setzt Datum, Wochentag und Uhrzeit in der Uhr. Angaben zum Format siehe bei GET_DATE_AND_TIME.

Beispiel: Die Uhr soll auf Freitag, den 21.12.2001 und 12:58:00 gestellt werden (im 24-Stunden-Mode): eax = 140c3a00h, ebx = 010c1505

Entry:	eax	Jahrhundert ^ Stunden ^ Minuten ^ Sekunden	
	ebx	Jahr ^ Monat ^ Tag ^ Wochentag	
		(bei Wochentag: 0 = Sonntag, 1 = Montag ...)	
Changed:	f, eax, ebx		
Exit (CY=0):	-		
Exit (CY=1):	ax	Fehlercode:	20e1h = Device wird von OsX benutzt

GET_ALARM (Nr. 131)

Diese Subroutine liest die Alarmzeit aus der Uhr

Entry:	-		
Changed:	f, eax		
Exit (CY=0):	eax	Alarmzeit: 00 ^ Stunden ^ Minuten ^ Sekunden	
Exit (CY=1):	ax	Fehlercode:	22e1h = Device nicht zugreifbar
			21e1h = unerlaubte Zeitangabe von Device
			20e1h = Device wird von OsX benutzt

SET_ALARM (Nr. 132)

Diese Subroutine setzt die Alarmzeit in der Uhr

Entry:	eax	Zeit: 00 ^ Stunden ^ Minuten ^ Sekunden	
Changed:	f, eax		
Exit (CY=0):	-		
Exit (CY=1):	ax	Fehlercode:	20e1h = Device wird von OsX benutzt

READ_CMOS_RAM (Nr. 120)

Diese Subroutine liest einen Datenblock aus dem CMOS-RAM

Entry:	ax	Nr. des ersten Byte: 14..127 (= 0eh..7fh)	
	cx	Anzahl Byte: 0..114	
	ebx	Ziel-Adresse (physikalisch)	
Changed:	f, ax, ebx, cx, dx		
Exit (CY=0):	-		
Exit (CY=1):	ax	Fehlercode:	10e0h = Falscher Parameter bei Aufruf

WRITE_CMOS_RAM (Nr. 121)

Diese Subroutine schreibt einen Datenblock in das CMOS-RAM

Entry:	ax	Nr. des ersten Byte: 14..127 (= 0eh..7fh)	
	cx	Anzahl Byte: 0..114	
	ebx	Quell-Adresse (physikalisch)	
Changed:	f, ax, ebx, cx, dx		
Exit (CY=0):	-		
Exit (CY=1):	ax	Fehlercode:	10e0h = Falscher Parameter bei Aufruf

GET_TIMER (Nr. 49)

Diese Subroutine liefert den aktuellen Timer-Wert und den Status von Timer A, B oder C. Einzelheiten zu den Timern finden Sie in der Beschreibung des Bausteins 8254, z.B. von Intel.

Entry:	al	Timer: 0 = A, 1 = B, 2 = C	
Changed:	f, ax, cl		
Exit (CY=0):	ax	Aktueller Zählerstand	
	cl	Status:	
		Bit 0: 0 = binär, 1 = dezimal	
		Bit 1 bis 3: Mode	
		Bit 4 und 5: = 0	
		Bit 6: Null-Count	
		Bit 7: Zustand des Output-Pins	
Exit (CY=1):	ax	Fehlercode:	10e0h = Falscher Parameter bei Aufruf

SET_TIMER (Nr. 50)

Diese Subroutine setzt die Betriebsart (Mode) und den Zählerwert von Timer A, B oder C. Einzelheiten zu den Timern finden Sie in der Beschreibung des Bausteins 8254, z.B. von Intel. Für die üblichen Anwendungen wird der Timer in Mode 2 betrieben.

Um einen Timer anzuhalten, kann er vorübergehend z.B. in Mode 5 gesetzt werden. Er kann dann definiert gestartet werden, in dem er in Mode 2 oder Mode 3 gesetzt wird.

Entry:	al	Timer: 0 = A, 1 = B, 2 = C
	ah	Mode: 0, 1, 2, 3, 4 oder 5
	cx	Zählerwert
Changed:	f, ax	

Exit (CY=0):	-		
Exit (CY=1):	ax	Fehlercode:	10e0h: Falscher Parameter bei Aufruf

READ_EEPROM_BLOCK (Nr. 51)

READ_EEPROM_BLOCK liest einen Block aus dem EEPROM eines Moduls.

Das Auslesen direkt aus einem EEPROM ist nur in den seltensten Fällen erforderlich. Die Informationen aus den EEPROMs werden nach jedem Reset automatisch in den Parameterbereich des Betriebssystems übertragen und stehen dann dort für den Anwender zur Verfügung.

Entry:	ah	SLN (Slot^Layer-Nummer)	
	ebx	Ziel, wo die gelesenen Daten hin sollen	
	cx	Anzahl Byte	
	edx	Rel. EEPROM-Adresse	
Changed:	eax, dx		
Exit (CY=0):	ax	=0: kein Fehler	
Exit (CY=1):	ax	Fehlercode:	2ae1h = Modul nicht bereit
			80e1h = Kein Modul auf Steckplatz
			85e1h = Falscher EEPROM-Typ
			20e1h = Semaphore nicht bekommen
			2be1h = EEPROM nicht bereit
			14e1h = Kennung bei Lesen von Device falsch

WRITE_EEPROM_BLOCK (Nr. 52)

WRITE_EEPROM_BLOCK schreibt einen Block in ein EEPROM eines Moduls.

Entry:	ah	SLN (Slot^Layer-Nummer)	
	ebx	Pointer auf Daten	
	cx	Anzahl Byte	
	edx	Rel. EEPROM-Adresse	
Changed:	eax, ebx, cx, dx		
Exit (CY=0):	ax	=0: kein Fehler	
Exit (CY=1):	ax	Fehlercode:	2ae1h = Modul nicht bereit
			80e1h = Kein Modul auf Steckplatz
			85e1h = Falscher EEPROM-Typ
			20e1h = Semaphore nicht bekommen
			2be1h = EEPROM nicht bereit
			14e1h = Kennung bei Lesen von Device falsch

GET_TDT_ADDRESS (Nr. 54)

GET_TDT_ADDRESS liefert die physikalische Adresse des Anfangs der Task-Deskriptor-Tabelle (TDT) einer Task. Aus Geschwindigkeitsgründen findet bei dieser Subroutine ausnahmsweise keine Fehlerprüfung statt (das CY-Flag ist nach Verlassen der Subroutine immer = 0 gesetzt), es wird also nicht geprüft, ob unter der Task ein Programm installiert ist. Diese Überprüfung muß im aufrufenden Programm selbst vorgenommen werden (z.B. durch Prüfung der Flags in der TDT). Alternativ kann auch die Subroutine GET_TASK_INFO verwendet werden, die eine Fehlerprüfung beinhaltet, aber langsamer ist.

Entry:	eax	Tasknummer
Changed:	f, eax	
Exit (CY=0):	eax	32 Bit physikal. Adresse der TDT

GET_INT_TASK (Nr. 58)

GET_INT_TASK ermittelt Nummer und Typ der Task, die einen bestimmten Interrupt nutzt.

Entry:	al	Interrupt-Nr. (0 bis 255)
Changed:	f, ax, bx	
Exit (CY=0):	al	Tasktyp 0 = NI-Task, 1 = II-Task, 2 = DI-Task, 3 = TI-Task, 7 = Systemtask, z.B. Taskmanager
	bx	Nr. der Task, die den Interrupt nutzt (bx = - 1, wenn der Interrupt nicht benutzt ist)
Exit (CY=1):	ax	Fehlercode: 0bd7h = "Warnung vom System", z.B. wenn die betriebssysteminternen Tabellen Unsinn enthalten (= Absturz des Systems).

CONVERT_TIMER_DATA (Nr. 59)

Diese Subroutine wandelt eine Zeit, die in Mikrosekunden angegeben wird, in den passenden Timer-Wert zum Programmieren eines Timers (A, B oder C), z.B. für die Verwendung mit der Subroutine SET_TIMER (Nr. 50). Dadurch kann die Programmierung in absoluten Zeitwerten erfolgen, unabhängig von der Frequenz eines Quarzoszillators auf dem MAX-PC, der den Eingangstakt für die Timer liefert.

Entry:	eax	Zeit (in Mikrosekunden)
Changed:	f, eax	
Exit (CY=0):	ax	Timer-Wert für Timer A, B oder C
Exit (CY=1):	ax	Fehlercode: 21e0h = "unerlaubte Zeitangabe"

SEND_HOST_SRQ (Nr. 53)

SEND_HOST_SRQ sendet ein Wort als Service Request zum Host. Auf dem Host wird dadurch ein Interrupt ausgelöst, sofern ein Interrupt-Kanal gewählt ist. Diese Subroutine darf nur aus der Hauptprozedur von Nicht-Interrupt-Tasks (NI-Task) aufgerufen werden, nicht aus Interrupt-Tasks (II-Task) oder Timer-Initiierten Tasks (TI-Tasks). Wenn Sie statt SEND_HOST_SRQ die Routine SEND_BUFFER_SRQ (Nr. 60) verwenden, gibt es diese Einschränkung nicht.

Auch Fehlermeldungen des Betriebssystems werden auf diese Weise zum Host (zunächst zum PC, bei MAX6isa und MAX6pci immer Steckplatz-Nummer SLN=0) gemeldet. Dabei wird im Low Byte des SRQ-Wortes die Fehlergruppe (c0h bis ffh) gemeldet, siehe Anhang B des MAX6pci-Handbuchs.

Entry:	bx	SRQ-Wort
	ah	Slot^Layer-Nr des Ziels
Changed:	f, eax, cx, dx	
Exit (CY=0):	-	ok, gesendet
Exit (CY=1):	al	0d4h: Fehlermeldung "SRQ nicht gesendet"
	ah	Grund: Bit 15: X-Bus Time-Out Bit 9 = 1: Empfänger Mailbox voll Bit 8 = 1: Mailbox Semaphore Empfänger besetzt

SEND_BUFFER_SRQ (Nr. 60)

SEND_BUFFER_SRQ sendet ein Wort über einen Puffer (FIFO-Prinzip) als Service-Request an einen Host. Diese Subroutine kann aus allen Tasks heraus aufgerufen werden.

Die Größe des Puffers (in Anzahl Byte) kann vor dem ersten Aufruf dieser Routine in Systemparameter 202 (Wort) angegeben werden. Nach dem ersten Aufruf steht dort die tatsächliche Größe des Puffers und in Systemparameter 204 (Wort) die Puffernummer. Die Länge eines SRQ bzw. einer Fehlermeldung im Puffer beträgt 4 Byte.

Entry:	ah	Protokoll: 0: al enthält Host-Nr. 1: Inter-MAX-Kommunikation, al enthält SLN (Slot^Layer-Nr.) des Ziel ≥2: SRQ wird verworfen
	al	Bei Protokoll = 0: al = 0 : Reserviert al = 1: Inter-MAX-Kommunikation mit SLN = 0 (=Basiskarte PC) al = 2: COM2 serielle Schnittstelle des X-MAX-X al = 3..15: Host-Nr. (Anwenderdefiniert) Bei Protokoll = 1: Inter-MAX-Kommunikation, al = SLN
	bx	Das zu sendende SRQ-Wort: Im Low Byte ist für Anwender-SRQs nur 80h bis bfh erlaubt, im High Byte 0 bis ffh.
	cx	Reserviert (immer = 0 setzen)
	dx	Reserviert (immer = 0 setzen)
Changed:	f, eax, cx	

Exit (CY=0):	-	ok, SRQ in Puffer eingetragen
Exit (CY=1):	ax	Fehlercode bzw. Warnung:
		22d7h = "Puffer wird gerade beschrieben"
		24d7h = "Nicht genug Platz im Puffer"
		26e0h = "Puffer-Nr. ungültig"
		90e0h = "Host schon/noch installiert"
		91e0h = "Host nicht installiert"
		10e0h = Falscher Parameter bei Aufruf der Subroutine, z.B. "Host-Nr. nicht erlaubt"

MACRO_LOCKING (Nr. 61)

Es gibt bei Makro-Befehlen drei Phasen bzgl. der Unterbrechbarkeit:
1. Wen ein Makro-Befehl unterbrechen kann
2. Von wem ein laufender Makro-Befehl unterbrochen werden kann
3. Was in den Pausen zwischen Makro-Befehlen passieren darf

Standardmäßig sind die Makro-Befehle nicht unterbrechbar (*=Standardeinstellung):

*	Bit 0:	Makro darf NI-Task unterbrechen
	Bit 1:	Makro darf TI-Task unterbrechen
	Bit 2:	Makro darf II- und DI-Task unterbrechen
	Bit 3:	Makros durch TI-Task unterbrechbar
	Bit 4:	Makros durch II- oder DI-Task unterbrechbar
*	Bit 5:	Pause zwischen Makros durch NI-Tasks benutzbar
*	Bit 6:	Pause zwischen Makros durch TI-Tasks benutzbar
*	Bit 7:	Pause zwischen Makros durch II- oder DI-Task benutzbar

Bisher ist nur Bit 4 realisiert. Hierfuer wird der Interrupt-Vektor für die PC-Schnittstelle verbogen. Diese Funktion/Befehl kann jederzeit aufgerufen werden.

Entry:	al	f = Flags (0=nein, 1=ja),
	ah	m = Maske (0=nicht ändern, 1=ändern)
Changed:	f, ax	
Exit (CY=0):	-	kein Fehler
Exit (CY=1):	ax	Fehler, z.Zt. keiner definiert

WAKEUP_TI_TASK (Nr. 65)

WAKEUP_TI_TASK aktiviert eine TI-Task (timerinitiierte Task) mit einem als Parameter übergebenen Zeitplan. Er beschreibt, wann und wie oft die TI-Task aufgerufen wird (siehe Beschreibung Kapitel 5 des MAX6pci-Handbuchs). Basis des Zeitplans für alle TI-Tasks ist ein sogenannter Timer-Tick, wofür standardmäßig Timer C mit 1 ms Takt verwendet wird. Die Taktrate kann geändert werden, indem vor der ersten Installierung einer TI-Task der Parameter 316 des Betriebssystems geändert wird (Doppelwort, Zeitangabe in µs).

Wenn eine bereits aktivierte TI-TASK noch einmal mit WAKEUP_TI_TASK aktiviert wird, werden die alten Parameter verworfen und die neu übergebenen verwendet.

WAKEUP_TI_TASK kann jederzeit aus allen Tasks heraus aufgerufen werden. Eine laufende TI-Task kann sich auch selbst neu aktivieren oder deaktivieren und damit ihren eigenen Zeitplan ändern.

Um einer TI-Task eine sehr hohe Priorität einzuräumen, müssen Priorität und Hold-Off-Zeit = 0 gesetzt werden. Die übrigen Parameter können wie gewünscht eingestellt werden. Sie kommt dann auf jeden Fall als nächste TI-Task (beim nächsten Timer-Tick) an die Reihe. Ein gerade laufender Aufruf einer TI-Task wird aber zunächst normal beendet.

Benötigt eine TI-Task mehr Zeit als zwischen zwei Timer-Ticks zur Verfügung steht, dann sorgt das Betriebssystem dafür, daß die "verlorene" Zeit wieder aufgeholt wird, also daß die nachfolgenden TI-Tasks wieder zum vorgesehenen Zeitpunkt aufgerufen werden. Wann das erreicht ist, hängt von der aktuellen CPU-Auslastung ab.

Entry:	dx	Tasknummer
	cl	Ordnungszahl für Priorität (cl = 0: höchste, cl = 255: niedrigste). Bei gleicher Ordnungszahl erhält die zuletzt aufgerufene TI-Task die höhere Priorität.
	ch	= 0 (für zukünftige Entwicklungen reserviert)
	eax	Anzahl von Aufrufen, nach denen die Task wieder deaktiviert wird (eax = 0: unendlich)
	esi	Intervall zwischen zwei Aufrufen der TI-Task in Anzahl Timer-Ticks
	edi	Hold-Off-Zeit: Verzögerung vom Aktivieren bis zum ersten Aufruf der Task, in Anzahl Timer-Ticks
Changed:	f, eax, ebx, ecx, edx, esi, edi	
Exit (CY=0):	-	-
Exit (CY=1):	ax	Fehlercode: 08e0h = "kein Programm installiert"

CREATE_BUFFER (Nr. 70)

Einige wichtige Punkte beim Arbeiten mit diesen Puffern sind in Kapitel 5 des MAX6pci-Handbuchs beschrieben. CREATE_BUFFER legt einen Ringpuffer im Speicher auf der Karte an und liefert eine Puffernummer, die für alle weiteren Zugriffe auf den Puffer benutzt wird. Es können max. 256 voneinander unabhängige Puffer angelegt werden. Nach dem Aufruf von CREATE_BUFFER ist der Puffer leer. Beim Anlegen eines Puffers sind für das Reservieren von Speicherplatz verschiedene **Strategien** möglich:

1. *UP absolut:* Der Puffer soll so groß sein wie angegeben und von der unteren Grenze des freien RAM an aufwärts reserviert werden. Wenn nicht genug Platz ist, wird kein Puffer angelegt.
2. *UP max.:* Der Puffer soll so groß wie möglich, aber nicht größer als angegeben sein und von der unteren Grenze des freien RAM an aufwärts reserviert werden.
3. *DOWN absolut:* Der Puffer soll so groß sein wie angegeben und von der oberen Grenze des freien RAM an abwärts reserviert werden. Wenn nicht genug Platz ist, wird kein Puffer angelegt.
4. *DOWN max.:* Der Puffer soll so groß wie möglich, aber nicht größer als angegeben sein und von der oberen Grenze des freien RAM an abwärts reserviert werden.

Alignment:
Hiermit kann die Anfangsadresse des Puffers so gewählt werden, daß sie ohne Rest durch n teilbar ist, wobei n = 0, 1, 2, 4, 8, 16, ... sein kann.

Die Angaben der Tasknummer und der Art der Verwendung des Puffers dienen betriebssysteminternen Zwecken und können auch = 0 gesetzt werden. Sie sind für zukünftige Entwicklungen vorgesehen.

Entry:	eax	Absolute bzw. maximale Größe des anzulegenden Puffers (in Anzahl Byte)
	bl	Strategie: 1: UP absolut 2: UP max. 3: DOWN absolut 4: DOWN max.
	bh	Byte-Alignment: n = 2**bh (bh max. <= 16) z.B.: 2^0= 1, 2^1 = 2, 2^2 = 4
	cx	Verwendung des Speichers (z. Zt. = 0)
	dx	Nr. der Task, die den Speicher nutzt (z. Zt. = 0)
Changed:	f, eax, ebx	
Exit (CY=0):	eax	Puffer-Nr.
	ebx	Größe des angelegten Puffers (in Anzahl Byte)
Exit (CY=1):	ax	Fehlercode: 13d7h = "Nicht genug Speicherplatz im RAM" 10e0h = "Falsche/r Parameter bei Aufruf"

DELETE_BUFFER (Nr. 71)

DELETE_BUFFER ist noch nicht implementiert.

Entry:	eax	Puffer-Nr.	
Changed:	f, eax		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	0fe0h = "Nicht implementiert"

CLEAR_BUFFER (Nr. 72)

CLEAR_BUFFER löscht den Inhalt eines Puffers. Der Puffer ist danach leer. Das Löschen entspricht logisch einem Lesezugriff, bei dem der gesamte Pufferinhalt ausgelesen wird.

Entry:	eax	Puffer-Nr.	
Changed:	f, eax		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig" 23d7h = "Puffer wird gerade gelesen"

GET_BUFFER_STATUS (Nr. 73)

GET_BUFFER_STATUS liefert die Anzahl gültiger Zeichen (Byte) und den freien Platz im Puffer. Beide Angaben addiert ergeben die Länge des Puffers. Die Routine kann unabhängig vom Lesen und Schreiben desselben Puffers durch andere Tasks aufgerufen werden.

Sie müssen diese Routine nicht vor jedem Schreiben in oder Lesen aus einem Puffer aufrufen. Es ist meistens einfacher (und auch genauso schnell), die entsprechende Schreib- oder Leseroutine direkt mit der gewünschten Blockgröße aufzurufen. Wenn nicht genug Platz ist bzw. nicht genug Zeichen im Puffer sind, erhalten Sie eine Fehlermeldung bzw. eine Rückmeldung und können entsprechend reagieren. **!**

Entry:	eax	Puffer-Nr.	
Changed:	f, eax, ebx		
Exit (CY=0):	eax	Anzahl gültiger Zeichen (Byte) im Puffer	
	ebx	Freier Platz im Puffer (in Anzahl Byte)	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"

WRITE_BUFFER_BYTE (Nr. 74)

WRITE_BUFFER_WORD (Nr. 75)

WRITE_BUFFER_DWORD (Nr. 76)

WRITE_BUFFER_BYTE schreibt ein Byte, WRITE_BUFFER_WORD ein Wort (2 Byte) und WRITE_BUFFER_DWORD ein Doppelwort (4 Byte) in den Puffer (siehe auch WRITE_BUFFER_BLOCK und WRITE_BUFFER_MAX). Wenn nicht genug Platz im Puffer ist, wird nichts geschrieben und es erfolgt eine Fehlermeldung.

Entry:	eax	Puffer-Nr.	
	bl	Datenbyte (bei WRITE_BUFFER_BYTE)	
	bx	Datenwort (bei WRITE_BUFFER_WORD)	
	ebx	Datendoppelwort (bei WRITE_BUFFER_DWORD)	
Changed:	f, eax		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			24d7h = "nicht genug Platz im Puffer"
			22d7h = "P. wird gerade beschrieben"

WRITE_BUFFER_BLOCK (Nr. 77)

WRITE_BUFFER_BLOCK schreibt eine Anzahl Zeichen (Byte) in den Puffer. Dabei wird nach dem "Alles-oder-nichts-Prinzip" verfahren, d. h. es werden alle Zeichen in den Puffer geschrieben, wenn genügend Platz im Puffer ist, oder gar nichts (siehe auch WRITE_BUFFER_BYTE und WRITE_BUFFER_MAX).

Entry:	eax	Puffer-Nr.	
	ebx	Adresse (physikal.) der zu schreibenden Daten	
	ecx	Anzahl Byte zu schreiben (0 bis 65520)	
Changed:	f, eax, ebx, ecx		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			24d7h = "nicht genug Platz im Puffer"
			22d7h = "P. wird gerade beschrieben"

WRITE_BUFFER_MAX (Nr. 78)

WRITE_BUFFER_MAX schreibt eine Anzahl Zeichen (Byte) in den Puffer. Dabei wird nach dem "Soviel-wie-möglich-Prinzip" verfahren, d. h. es werden soviel Zeichen in den Puffer geschrieben wie Platz im Puffer ist, maximal aber nur soviel wie beim Aufruf angegeben. Wenn nicht genügend Platz im Puffer ist, wird der Puffer voll geschrieben und die Anzahl Zeichen, die nicht im Puffer untergebracht werden konnten, zurückgemeldet (siehe auch WRITE_BUFFER_BLOCK).

Entry:	eax	Puffer-Nr.	
	ebx	Anfangsadresse (physikal.) der in den Puffer zu schreibenden Daten	
	ecx	Anzahl Byte zu schreiben (0 bis 65520)	
Changed:	f, eax, ebx, ecx		
Exit (CY=0):	eax	Rest = Anzahl Byte, die nicht in den Puffer geschrieben werden konnten.	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			22d7h = "P. wird gerade beschrieben"

READ_BUFFER_BYTE (Nr. 79)
READ_BUFFER_WORD (Nr. 80)
READ_BUFFER_DWORD (Nr. 81)

READ_BUFFER_BYTE liest ein Byte, READ_BUFFER_WORD ein Wort (2 Byte) und READ_BUFFER_DWORD ein Doppelwort (4 Byte) aus dem Puffer (siehe auch READ_BUFFER_BLOCK, READ_BUFFER_MAX, VIEW_BUFFER_BLOCK und VIEW_BUFFER_-MAX). Wenn nicht genug Zeichen im Puffer sind, wird nichts gelesen und es erfolgt eine Fehlermeldung.

Entry:	eax	Puffer-Nr.	
Changed:	f, eax		
Exit (CY=0):	al	Datenbyte (bei READ_BUFFER_BYTE)	
	ax	Datenwort (bei READ_BUFFER_WORD)	
	eax	Datendoppelwort (bei READ_BUFFER_DWORD)	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			25d7h = "nicht genug Zeichen im P."
			23d7h = "P. wird gerade ausgelesen"

READ_BUFFER_BLOCK (Nr. 82)

READ_BUFFER_BLOCK liest Zeichen (Byte) aus dem Puffer. Dabei wird nach dem "Alles-oder-nichts-Prinzip" verfahren, d. h. es wird die gewünschte Anzahl Zeichen aus dem Puffer gelesen, sofern genügend Zeichen im Puffer sind, oder gar nichts (siehe auch READ_BUFFER_BYTE, READ_BUFFER_MAX, VIEW_BUFFER_ BLOCK und VIEW_-BUFFER_MAX).

Entry:	eax	Puffer-Nr.	
	ebx	Adresse (32 Bit physikal.), wo die zu lesenden Daten hin sollen	
	ecx	Anzahl Byte zu lesen (0 bis 65520)	
Changed:	f, eax, ebx, ecx		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			25d7h = "nicht genug Zeichen im P."
			23d7h = "P. wird gerade ausgelesen"

READ_BUFFER_MAX (Nr. 83)

READ_BUFFER_MAX liest eine Anzahl Zeichen (Byte) aus dem Puffer. Dabei wird nach dem "Soviel-wie-möglich-Prinzip" verfahren, d. h. es werden soviel Zeichen aus dem Puffer gelesen wie gewünscht, maximal aber nur soviel wie beim Aufruf angegeben. Wenn nicht so viele Zeichen wie gewünscht im Puffer sind, wird der Puffer leer gelesen und die Anzahl Zeichen, die nicht gelesen werden konnten, zurückgemeldet (siehe auch READ_BUFFER_BYTE, READ_BUFFER_BLOCK, VIEW_BUFFER_BLOCK und VIEW_BUFFER_MAX).

Entry:	eax	Puffer-Nr.	
	ebx	Anfangsadresse (32 Bit physikal.), wo die aus dem Puffer zu lesenden Daten hin sollen	
	ecx	Anzahl Byte zu lesen (0 bis 65520)	
Changed:	f, eax, ebx, ecx		
Exit (CY=0):	eax	Rest = Anzahl Byte, die nicht aus dem Puffer gelesen werden konnten.	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			23d7h = "P. wird gerade ausgelesen"

VIEW_BUFFER_BLOCK (Nr. 84)

VIEW_BUFFER_BLOCK kopiert eine Anzahl Zeichen (Byte) aus dem Puffer, die Zeichen bleiben aber im Puffer unverändert enthalten.

Dabei wird nach dem "Alles-oder-nichts-Prinzip" verfahren, d. h. es wird die gewünschte Anzahl Zeichen aus dem Puffer kopiert, sofern genügend Zeichen im Puffer sind, oder gar nichts (siehe auch VIEW_BUFFER_MAX, READ_BUFFER_BYTE, READ_BUFFER_BLOCK und READ_BUFFER_MAX).

Entry:	eax	Puffer-Nr.	
	ebx	Anfangsadresse (32 Bit physikal.), wo die zu kopierenden Zeichen hin sollen	
	ecx	Anzahl Byte zu kopieren (0 bis 65520)	
Changed:	f, eax, ebx, ecx		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			25d7h = "nicht genug Zeichen"
			23d7h = "P. wird gerade ausgelesen"

VIEW_BUFFER_MAX (Nr. 85)

VIEW_BUFFER_MAX kopiert eine Anzahl Zeichen (Byte) aus dem Puffer, die Zeichen bleiben aber im Puffer unverändert enthalten.

Dabei wird nach dem "Soviel-wie-möglich-Prinzip" verfahren, d. h. es werden soviel Zeichen aus dem Puffer kopiert wie gewünscht, maximal aber nur soviel wie beim Aufruf angegeben. Wenn nicht so viele Zeichen im Puffer sind wie angegeben, werden alle Zeichen, die im Puffer sind, kopiert, und die Anzahl Zeichen, die nicht kopiert werden konnten, zurückgemeldet (siehe auch VIEW_BUFFER_BLOCK, READ_BUFFER_BLOCK und READ_BUFFER_MAX).

Entry:	eax	Puffer-Nr.
	ebx	Anfangsadresse (32 Bit physikal.), wo die aus dem Puffer zu kopierenden Daten hin sollen
	ecx	Anzahl Byte zu kopieren (0 bis 65520)
Changed:	f, eax, ebx, ecx	
Exit (CY=0):	eax	Rest = Anzahl Byte, die nicht aus dem Puffer kopiert werden konnten
Exit (CY=1):	ax	Fehlercode: 26e0h = "Puffer-Nr. ungültig" 23d7h = "P. wird gerade ausgelesen"

GET_TASK_NUMBER (Nr. 90)

Mit GET_TASK_NUMBER kann die Nummer einer Task ermittelt werden, unter der ein Programm installiert ist. In den meisten Anwendungsfällen ist ein Programm nur unter einer Task installiert. Es kann aber auch mehrmals unter verschiedenen Tasks installiert sein (der Programmcode muß aber nur einmal auf der Karte sein). Mit dieser Systemroutine können sowohl die Anzahl der Installierungen als auch alle zugehörigen Tasknummern ermittelt werden.

Entry:	ax	Programm-Nr.
	cx	Aufrufparameter n n = 1: melde die niedrigste Tasknummer, unter der das Programm installiert ist n > 1: melde die jeweils höhere Tasknummer, falls das Programm mindestens n mal installiert ist n = 0 oder -1 (= ffffh): melde die höchste Tasknummer, unter der das Programm installiert ist und die Anzahl der Installierungen
Changed:	f, ax, ebx, cx, dx, esi, di	
Exit (CY=0):	ax	Programm-Nr. (unverändert)
	cx	cx = 0 (für alle n): Programm ist nicht installiert, dx ungültig cx = n (n = 1 bis 1024): Programm ist mindestens n-mal installiert, dx = Tasknummer mit der n-höchsten Tasknummer aller Installierungen dieses Programmes. cx = 1 bis 1024 (n = -1): cx = Anzahl Installierungen, dx = höchste Tasknummer aller Installierungen dieses Programmes.

		cx < n (n = 2 bis 1024): Annahme, daß das Programm n-mal installiert ist, war falsch, das Programm ist nur cx-mal installiert. dx = Tasknummer mit der cx-höchsten Tasknummer aller Installierungen dieses Programmes.
	dx	Tasknummer, sofern gültig (s.o.)
Exit (CY=1):	ax	Fehler, z. Zt. kein Fehler definiert

Beispiel 1: Prüfen, ob und wie oft ein Programm installiert ist:

Entry:	cx = 0	
Exit (CY=0):	cx = 0:	Programm ist nicht installiert
	cx > 0:	Programm ist cx mal installiert, z.B.:
		cx = 1: Programm ist 1 mal installiert, dx = Tasknummer dieser Installierung
		cx = 2: Programm ist 2 mal installiert, dx = Tasknummer der Installierung mit der höchsten Tasknummer. Um die andere Tasknummer zu bekommen, muß diese Subroutine noch einmal mit cx = 1 aufgerufen werden.
		cx = 3: Programm ist 3 mal installiert, dx = Tasknummer der Installierung mit der höchsten Tasknummer. aller 3 Installierungen. Um die anderen beiden Tasknummern zu bekommen, muß diese Subroutine noch 2 mal aufgerufen werden, mit cx = 1 und dann mit cx = 2 (ax ist nach dem Aufruf nicht verändert).

Beispiel 2 (Sonderfall): Wenn man weiß, daß ein Programm nur einmal installiert sein kann, kann man Zeit sparen:

Entry:	cx = 1	
Exit (CY=0):	cx = 0:	Programm ist nicht installiert.
	cx = 1:	Programm ist 1 mal installiert, dx = Tasknummer.

Beispiel 3 (Sonderfall): Wenn ein Programm mehrfach installiert ist, sind alle Tasknummern verschieden. Mit cx wird die Ordnungszahl der Tasknummer angegeben, die man haben möchte:

cx = 1: niedrigste,
cx = 2: zweitniedrigste,
cx = 3: drittniedrigste, etc. bis
cx = -1 (ffffh): höchste Tasknummer

Fall 3a)

Entry:	cx = 1	
Exit (CY=0):	cx = 0:	Programm ist nicht installiert, dx ist ungültig
	cx = 1:	Programm ist 1 mal installiert, dx = Tasknummer

Fall 3b)

Entry:	cx = 2	
Exit (CY=0):	cx = 0:	Programm ist nicht installiert., dx = ungültig
	cx = 1:	Programm ist nur 1 mal installiert, dx = Tasknummer dieser Installierung
	cx = 2:	Programm ist 2 mal installiert, dx = Tasknummer der Installierung mit der höchsten Tasknummer

INIT_IO (Nr. 93)

INIT_IO initialisiert ein Modul entsprechend den im zugehörigen EEPROM angegebenen Werten.

Entry:	al	SLN = Modulsteckplatz (= Slot^Layer-Nummer)
	ah	ah = 0: nur initialisieren, wenn Bit 0 in WORT-1 des zugehörigen EEPROMs = 1 ist ah = 1: in jedem Fall initialisieren
Changed:	f, ax	
Exit (CY=0):	ax	ax = 0: ok, Initialisierung erfolgt
Exit (CY=1):	ax	Modul wurde nicht initialisiert wegen:
		Fehlercode: 1be0h: "Falscher Modulsteckplatz" 10e0h: "Falscher Parameter bei Aufruf" 1ce0h: "EEPROM-Info falsch" 15e1h: "Device nicht vorhanden"
		Warnungen: 80d4h: "Kein Modul auf dem Steckpl." 80d4h: "Modul braucht keine Initialis." 82d4h: "Bit-0 in EEPROM WORT-1=0 83d4h: "Keine Init-Subroutine in OsX"

Folgende Module können z.Zt. (ab Betriebssystemversion " MAX-11A.01x ") initialisiert werden:

Typ	Modul	Typ	Modul
11	X-MAX-1	48	X-DA12-4
16	X-REL-8	50	X-DA16-4
24	X-OPT-200	54	X-C16-3
25	X-OPT-164	58	X-CAN-2
26	X-OPT-128	64	X-IDE-1
27	X-OPT-812	66	X-SSI-8
28	X-OPT-416		X-IEC-1
29	X-OPT-020		X_COM-2
32	X-AD16-4-		X-COM-8
36	X-AD14-20		X-ETH-4C
40	X-DIO-40		X-ETH-10
41	X-DIO-32		X-COM-4
44	X-DAD-4		

GET_INT_VECTOR (Nr. 91)

Diese Systemroutine liest den Interrupt-Vektor.

Entry:	al	Interrupt-Nr. (0 - 255 erlaubt) (Int-Nr. und Vektor-Nr. sind identisch)
Changed:	f, bx, cx, dx	
Exit (CY=0):	-	kein Fehler
	dx:cx	Vector (Segment:Offset)
Exit (CY=1):	ax	Fehler: 10e0h: "falscher Parameter bei Aufruf"

SET_INT_VECTOR (Nr. 92)

Diese Systemroutine setzt den Interrupt-Vektor.

Entry:	al	Interrupt-Nr. (0 - 255 erlaubt) (Interrupt-Nr. und Vektor-Nr. sind identisch)
	dx:cx	Vector (Segment:Offset)
Changed:	f, ax, bx, cx	
Exit (CY=0):	-	kein Fehler
Exit (CY=1):	ax	Fehlercode: 10e0h = "falscher Parameter bei Aufruf"

INSTALL_HOST (Nr. 96)

Installiere Host

Entry:	al	Host-Nr.
	ah	reserviert (=0 setzen)
	dx	Task-Nr. des neuen Host
	bx	Nr. der Funktion der Host-Task, die von OsX aufgerufen werden muß, um einen SRQ an den Host zu senden
	cx	Nr. der Funktion der Host-Task, die von einer anderen Task aufgerufen werden kann, und mit der sie sich für Benachrichtigung im Fehlerfall dort anmeldet
Changed:	f, ax, bx	
Exit (CY=0):	ax	kein Fehler, Ergebnis
Exit (CY=1):	ax	Fehlercode: 10e0h „ falscher Parameter bei Subroutine“ 90e0h = „Host schon installiert“

UNINSTALL_HOST (Nr. 97)

Die Host-Nr. in der Tabelle wird = -1 gesetzt. Derjenige, der UNINSTALL_HOST aufruft, muß dafür sorgen, daß die Anmeldungnen für Fehlerbehandlung bei sich gelöscht werden. Wenn noch ein SRQ im Puffer für einen gerade abgemeldeten Host sein sollte, wird er verworfen.

Entry:	al	Host-Nr.
	ah	reserviert (=0 setzen)
Changed:	f, ax, bx	
Exit (CY=0):	ax	kein Fehler, Ergebnis
Exit (CY=1):	ax	Fehlercode: 10e0h „ falscher Parameter bei Subroutine“ 91e0h = „Host war nicht installiert“

GET_HOST_INFO (Nr. 98)

Lies Information über Host.

Entry:	al	Host-Nr.
	ah	reserviert (=0 setzen)
Changed:	f, ax, bx	
Exit (CY=0):	ax	Host-Nr.
	dx	Task-Nr. des neuen Host
	bx	Nr. der Funktion der Host-Task, die von OsX aufgerufen werden muß, um einen SRQ an den Host zu senden
	cx	Nr. der Funktion der Host-Task, die von einer anderen Task aufgerufen werden kann, und mit der sie sich für Benachrichtigung im Fehlerfall dort anmeldet
Exit (CY=1):	ax	Fehlercode: 10e0h „falscher Parameter bei Subroutine“ 91e0h = „Host war nicht installiert“

CALL_CMD (Nr. 99)

Jeder Befehl CMDxx wird als Subroutine mit CALL NEAR aufgerufen und muß mit RET NEAR verlassen werden: Es sind alle Bytes des Befehls empfangen worden, die stehen im RCV-BUFFER ab [fs:eax].
Bei Exit muss CMD-Code und Format-Code dekodiert werden:

f = 0 kennzeichnet Sonderfälle, z.Zt. 3:

a) wenn CMD-Code = 24h, dann wird 224h als Wort zum Absender zurückgeschickt und dann die Programmkontrolle an die physikal. Adresse DWort [ebx+2] übergeben.

b) wenn CMD-Code > 7fh, dann war es ein SRQ. Es wird keine Antwort an den Absender zurückgeschickt, aber die eigene Semaphore wieder freigegeben

c) es wird ein Error zurueckgemeldet, der als Wort in [ebx+2] steht. Die Länge ist = 4.

f = 1: die Länge der Antwort folgt als Wort ab [ebx+2]

f > 1: die Länge der Antwort = f

Entry:	eax	Anfangsadresse, wo Befehl steht
	ebx	Anfangsadresse, wo Antwort hin soll
	[ebx]	CMD-Code (nur das erste Byte)
	fs,gs	Big Real Segmente
	ds	0 (mit Real Segment)
Changed:	f,eax,cx,dx,si,di (+ evtl. in Macro-CMD)	
Exit (CY=x):	[ebx]	CMD-Code
	[ebx+1]	Format-Code f (Byte): f = 0: Sonderfall CMD24, SRQ oder ERROR f = 1: Länge = [WORD ebx+2] f > 1: Länge = f
	ds	unverändert (= 0)
	ebx	unverändert

MOVE_MEMORY_BLOCK (Nr. 103)

Übertrage Datenblock im Speicher von Quelle zum Ziel.

Entry:	cx	Blockgröße (Anzahl Byte, 0 bis 65535)
	eax	Pointer auf Quell-Daten
	ebx	Pointer auf Ziel-Daten
Changed:	eax, ebx, cx, edx, esi	
Exit (CY=0):	ax	-
Exit (CY=1):	ax	Fehlercode (s.u.)

FLASH_STATUS (Nr. 104)

Setze bzw. lies Status von Flash.

Entry:	bl	Dienst (Erkl. siehe Makro-Befehl FLASH_STATUS in Application Note 82)
	bh	Parameter
	cl	Steckplatz des Moduls, auf dem das Flash sitzt
	ch	IC-Nr. (bei X-MAX-1 = 3)
Changed:	f, eax, bx, cx, dx	
Exit (CY=0):	ax	kein Fehler, Ergebnis
Exit (CY=1):	ax	Fehlercode (s.u.)

FLASH_ERASE (Nr. 105)

Lösche einen oder mehrere Sektoren im Flash.

Entry:	eax	Erste Adresse (auf Flash-Anfang bezogen und auf Sektoranfang aligned)
	ebx	Letzte Adresse (auf Flash-Anfang bezogen und auf Sektorende aligned)
Changed:	f, eax, ebx	
Exit (CY=0):	al	Letzter Sektor, der gelöscht wurde
	ah	0
Exit (CY=1):	ax	Fehlercode

FLASH_PROGRAM (Nr. 106)

Programmiere Block im Flash.

Entry:	eax	Pointer auf Datenblock
	ebx	Erste Adresse im Flash (rel. Adresse)
	cx	Anzahl Byte zu programmieren (max. 65535)
Changed:	f, eax, ebx, cx, dx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode (s.u.)

FLASH_READ (Nr. 107)

Lies Block aus Flash.

Entry:	eax	Pointer, wo die Daten hin sollen
	ebx	Erste Adresse im Flash (rel. Adresse)
	cx	Anzahl Byte zu lesen (max. 65535)
	dl	SLN (Slot^Layer-Nummer) des angesprochenen Flash
	dh	IC-Nr. (bei MAX-PC, Version X-MAX-1 = 3)
Changed:	f, eax, ebx, cx, dx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode (s.u.)

GET_TASKS_INSTALLED (Nr. 119)

Melde alle installierten Tasks, je Task wird ein Bit zurückgeliefert (1 = installiert).

Entry:	ax	= 0: reserviert
	ebx	Pointer, wo Daten hin sollen: 1024 Bit = 128 Byte
Changed:	f, ax, ebx	
Exit (CY=0):	[ebx]	Data (128 Byte)
Exit (CY=1):	ax	Fehlercode (s.u.)

READ_INDEX_PORT (Nr. 124)

Lies Byte von Index-Port.

Entry:	al	Index
	dx	Port für Index setzen/lesen, dx+1 ist Data-Port
Changed:	f, ax	
Unchanged:	Index	
Exit (CY=0):	al	Data
Exit (CY=1):	ax	Fehlercode (s.u.)

WRITE_INDEX_PORT (Nr. 125)

Schreibe Byte an Index-Port. Maske in cl wird zum Byte, das vom Index-Port gelesen wurde, undiert und zu dem Datenbyte in ah geodert. Das Ergebnis wird dann an den Index-Port geschrieben.

Entry:	al	Index
	ah	Data
	dx	Port für Index setzen/lesen, dx+1 ist Data-Port
	cl	Maske
Changed:	f, ax	
Unchanged:	Index	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode (s.u.)

READ_KONFIG (Nr. 126)

Lies Wort von X-Bus Konfigurations-Register.

Entry:	ah	SLN (Slot^Layer-Nummer)
	al	Konfigurations-Register
Changed:	f, eax, cx, dx	
Exit (CY=0):	cx	Data
Exit (CY=1):	ax	Fehlercode
		80e1h = Kein Modul auf Steckplatz
		10e1h = Falscher Parameter

WRITE_KONFIG (Nr. 127)

Schreibe Wort an X-Bus Konfigurations-Register.

Entry:	ah	SLN (Slot^Layer-Nummer)
	al	Konfigurations-Register
	cx	Data
Changed:	f, eax, dx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode (s.u.)
		80e1h = Kein Modul auf Steckplatz
		10e1h = Falscher Parameter

1.2. Beispiel: LED-Blinkprogramm in Assembler

Zur Funktion des Programms X1P0380 (Programm 380h):

Mit Parameter 0 (Command/Status) des Programms wird das Programm und damit das Blinken der LED gestartet (Parameter 0 = 1 startet das Blinken). Danach wird Parameter 0 vom Programm selbst = 2 gesetzt und zeigt dadurch an, daß es läuft. Wenn Parameter 0 wieder = 0 gesetzt wird, stoppt das Programm und das Blinken hört auf. In diesem Zustand kann durch Setzen von Parameter 1 = 0 die LED aus- bzw. = 1 eingeschaltet werden. Den gleichen Effekt kann man auch durch Aufruf der Prozeduren 2 bzw. 3 dieses Programms erreichen.

Zur Programmierung:

Das Programm X1P0380 ist im IDEAL Mode in Borland Turbo-Assembler (Tiny Model) geschrieben. Der Turbo-Assembler muß mit folgenden Parametern aufgerufen werden:

/m3: Erlaube 3 Phasen um Vorwärts-Referenzen zu finden
/mv32: Maximale Länge von Symbolen auf 32 setzen
/q: Fürs Linken nicht benötigte OBJ-Records unterdrücken

Der Turbo-Linker muß mit folgenden Parametern aufgerufen werden:

/3: 32-Bit processing ermöglichen
/t: COM-Datei erzeugen

Das Programm X1P0380 ist möglichst einfach gehalten. Es soll die Programmierung einer einfachen NI-Task auf dem MAX-PC und deren Installierung zeigen. Zusätzlich sind die Prozeduren 2 und 3 vorgesehen, die mit dem eigentlichen Programm nichts zu tun haben, aber es sollte hier auch gezeigt werden, wie eine vom PC oder von einer anderen Task aufrufbare Prozedur aussehen kann.

Die Auto-Init Prozedur (Prozedur 1) sorgt hier für eine Initialisierung der Parameter. Beachten Sie aber, daß in der PDT alle Prozeduren fortlaufend durchnumeriert sind und keine Adresse fehlen darf (wenn noch weitere Prozeduren folgen). Wenn also die Auto-Init Prozedur nicht benötigt wird, muß in der PDT trotzdem eine gültige Adresse stehen. Es ist ratsam, dann auch eine Auto-Init Prozedur vorzusehen, die aber nichts macht, sie besteht also nur aus "retf". Die Auto-Init Prozedur kann wie alle anderen Prozeduren auch vom PC oder von einer anderen Task auf der Karte aufgerufen werden. Sie unterscheidet sich nicht von anderen Prozeduren, außer daß sie als Teil des Installierungsvorgangs zum Schluß der Installierung des Programms unter einer Task automatisch aufgerufen werden kann. Beachten Sie, daß die in der PDT angegebenen Adressen relative Adressen sind.

Nach dem Laden des Programms auf die Karte beim Installieren des Programms unter einer Task erfolgt eine Anpassung dieser Adressen (Relozierung). Dieses Programm hat keinen Datenbereich, der Parameterbereich wird lokal vom Programm selbst reserviert. In diesem Fall ist zu beachten, daß vor dem Parameterbereich immer Platz für die TDT frei gehalten werden muß (s.u.).

Zur Installierung:

Das Programm hat keinen Datenbereich, der Parameterbereich wird lokal vom Programm selbst reserviert. Beim Installieren wird auch der Task-Typ (0 = NI-Task) und das Programmformat angegeben.

Beispiel für Installierung dieses Programms mit SNW32 unter Task 10h und anschließender Aktivierung der Task und Start:

```
; Beispielinstallation: LED-Blinken als NI-Task (ohne Interrupt)
;
; Reset einer MAX-Trägerkarte
MAXRESET board=0
;
; MAX-PC anwählen und Slot-Layer-Nummer einstellen
MAXCONNECTCPU board=0 slot=1 layer=0
;
; Osx auf MAX-PC laden
MAXLOADOSX osx="ROM"
;
; Programmnummer 380h unter Tasknummer 10h installieren
MAXINST file="X1P0380.EXE" no=300 task=10 tasktype=MAX_NI_TASK irq=0
        level=0 datasize=0 autoinit
; Task aktivieren
;
MAXCMD 41 02 10 00;
; Parameter der Task 10h setzen
; PAR-0 => 1 = Starten
;
MAXPAR task=10 start=0 para=01
```

```
;-----------------------------------------------------------------------------
;     X1P0380.ASM
;-----------------------------------------------------------------------------
;     Programm 0380h: Blink LED als NI-Task
;-----------------------------------------------------------------------------
;
;-----------------------------------------------------------------------------
      TITLE X1P0380
;-----------------------------------------------------------------------------
      MODEL TINY
      .486
      .CODE
      IDEAL
;-----------------------------------------------------------------------------

VERS0380    equ   '1'         ; Programm-Version ASCII: 1234
REV0380     equ   'A'         ; Programm-Revision ASCII: ABCD
TYPPDT      equ   1           ; Typ der PDT (Konstante)
DEFLPDT     equ   48          ; Laenge des Vorspanns der PDT (Konst.)
DEFLTDT     equ   36          ; Laenge der TDT (Konstante)
LED_ON      equ   800h+41*4   ; System-Subroutine 41
LED_OFF     equ   800h+42*4   ; System-Subroutine 42

;-----------------------------------------------------------------------------
;     X1P0380: Programm-Deskriptor-Tabelle (PDT)
;-----------------------------------------------------------------------------
X1P0380:
      DB    TYPPDT            ; 0   ; Typ der PDT
      DB    DEFLPDT           ; 1   ; Laenge des Vorspanns der PDT
      DW    (X1P0380E - X1P0380 - DEFLPDT)/4  ; Anzahl Prozeduren
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DW    00380h            ; 4   ; Programm-Nr.
      DB    VERS0380          ; 6   ; Programm-Version
      DB    REV0380           ; 7   ; Programm-Revision
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DB    4                 ; 8   ; Prozessor-Typ (1=186/V20,4=486)
      DB    0                 ; 9   ; Co-Prozessor-Typ (0=kein, 4=486DX)
      DB    1                 ; 10  ; Programmiersprache (1=ASM)
      DB    1                 ; 11  ; Programm-Typ (0=System, 1=Anwender)
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DB    08h               ; 12  ; Flag (Bit 0 bis 7): (ja=0)
                              ;     ;   Bit 0-2: Task-Typ (NI-Task=000)
                              ;     ;   Bit 3: Task-Typ, Int-Nr. von Install?
                              ;     ;   Bit 4: Real-Mode Programm?
                              ;     ;   Bit 5: Caching von Code erlaubt?
                              ;     ;   Bit 6: - (=0 setzen)
                              ;     ;   Bit 7: Kein Hypertext im Programm?
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DB    4                 ; 13  ; Flag (Bit 8 bis 15): (ja=0)
                              ;     ;   Bit 8: Daten von OsX reservieren?
                              ;     ;   Bit 9: Groesse Daten variabel?
                              ;     ;   Bit 10: Parameter von OsX reservie
                              ;           ren?
                              ;     ;   Bit 11-15: reserviert
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DB    0                 ; 14  ; Interrupt-Nr. (entfaellt, hier = 0)
      DB    0                 ; 15  ; reserviert
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DD    0                 ; 16  ; Anfangs-Adresse Datenbereich
      DD    0                 ; 20  ; Groesse Datenbereich
      DD    0                 ; 24  ; Minimum Datenbereich
      DD    10000h                  ; 28  ; Maximum Datenbereich
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DW    OFFSET PAR0380    ; 32  ; Anfangs-Adresse Parameterbereich
      DW    0                 ; 34  ;
      DW    PAR0380E - PAR0380; 36  ; Groesse Parameterbereich
```

```
      DD    0                 ; 38  ; Adresse Hypertextbereich
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DW    0                 ; 42  ; reserviert
      DD    0                 ; 44  ; reserviert fuer SORCUS
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DW    OFFSET X1P0380_M  ; 48  ; Adresse der Main-Prozedur (Nr.0)
      DW    0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DW    OFFSET X1P0380_I  ; 52  ; Adresse der Auto-Init Prozedur (Nr.1)
      DW    0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DW    OFFSET X1P0380_2  ; 56  ; Adresse Prozedur LED_ON (Nr.2)
      DW    0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DW    OFFSET X1P0380_3  ; 60  ; Adresse Prozedur LED_OFF (Nr.3)
      DW    0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

X1P0380E:

;-----------------------------------------------------------------------------
;     Parameterbereich (ist Teil des Programms)
;-----------------------------------------------------------------------------
;     vor dem Parameterbereich muss Platz fuer die TDT freigehalten werden

      db    DEFLTDT dup (0)         ; Laenge TDT

;-----------------------------------------------------------------------------

PAR0380:                            ; Anfang Parameterbereich
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; Name            Init  rel.Ad.             Bedeutung
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
STATUS:     db    0     ; 0         ; Command/Status:
                                    ; Command: 0 = Stop
                                    ;          1 = Start Blinking
                                    ; Status:  2 = Programm laeuft
ZULED:      db    0     ; 1         ; Zustand der LED: 0 = off, 1 = on
RATE:       dw    40000 ; 2         ; Blink-Rate (ungefaehr)
COUNTER:    dw    1     ; 4         ; Counter: auf 1, damit sofort umgeschal
                                    ; tet wird
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
PAR0380E:                           ; Ende Parameterbereich
```

```
;-----------------------------------------------------------------------
;     Main-Prozedur (= Prozedur 0): Blink LED
;-----------------------------------------------------------------------
;     Vom Betriebssystem wird in dx bei Aufruf einer Prozedur (auch der
;     Main-Prozedur) immer die Task-Nr uebergeben, unter der das Programm
;     installiert ist, zu dem die Prozedur gehoert. Das wird aber hier nicht
;     weiter ausgewertet.

PROC X1P0380_M    FAR                     ; alle Prozeduren muessen FAR sein

            pusha                         ; es muessten nicht alle Register
            push  ds                      ; gerettet werden, sondern nur die,
            push  es                      ;   die auch verwendet sind
            mov   ax,cs                   ; Tiny-Model: ds auf Anfang setzen
            mov   ds,ax
            xor   ax,ax                   ; es=0 fuer Aufruf von Systemrouti
                                          ; nen
            mov   es,ax
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            mov   al,[BYTE ds:STATUS]     ; Status = 2 (Run) ?
            cmp   al,2
            jnz   P0380_X                 ; nein

            mov   cx,[WORD ds:COUNTER]    ; Programm laeuft
            dec   cx
            jnz   P0380_LP                ; LED nicht umschalten

            mov   al,[BYTE ds:ZULED]      ; LED umschalten
            xor   al,1
P0380_ZX:   mov   [BYTE ds:ZULED],al      ; neuen Zustand der LED merken
            jz    P0380_OFF
            call  [DWORD es:LED_ON]
            jmp   P0380_LD
P0380_OFF:  call  [DWORD es:LED_OFF]

P0380_LD:   mov   cx,[WORD ds:RATE] ; setze Zaehler neu
P0380_LP:   mov   [WORD ds:COUNTER],cx
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
P0380_END:  pop   es
            pop   ds
            popa
            ret                           ; daraus macht TASM "retf"

P0380_X:    cmp   al,3                    ; Status = 3 (Fehler) ?
            jae   P0380_END               ; ja, Ende
            and   al,al                   ; nein, Status = 0 (Programm Stop)?
            jnz   P0380_MX
            mov   al,[BYTE ds:ZULED]      ; Setze LED auf gewuenschten Status
            cmp   al, 0
            jmp   P0380_ZX

P0380_MX:   mov   [BYTE ds:STATUS],2      ; setze Status immer = 2
            jmp   P0380_END

ENDP  X1P0380_M
```

```
;-----------------------------------------------------------------------------
;     Auto-Init Prozedur (= Prozedur 1): Parameter auf Anfangs-Zustand
;-----------------------------------------------------------------------------
;     Vom Betriebssystem wird in dx bei Aufruf einer Prozedur immer die
;     Task-Nr uebergeben, das wird aber hier nicht ausgewertet.

PROC  X1P0380_I   FAR

            pusha
            push  ds
            mov   ax,cs                   ; Tiny-Model: ds auf Anfang setzen
            mov   ds,ax
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            mov   bx,OFFSET P0380_PI      ; Tabelle
            mov   di,[WORD ds:bx]         ; erster zu initialisierender Para
                                          ; meter
            add   di,OFFSET PAR0380
            mov   cx,[WORD ds:bx+2]       ; Anzahl Byte
            add   bx,4                    ; Pointer
X1P0380_I1: mov   al,[BYTE ds:bx]
            mov   [BYTE ds:di],al
            inc   di
            inc   bx
            dec   cx
            jnz   X1P0380_I1
P0380_I2:
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            pop   ds
            popa
            ret
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
P0380_PI:   dw    0                 ; erster zu initialisierender Parameter
            dw    6                 ; Anzahl (Byte) Parameter zu initialisie
                                    ; ren
            db    0                 ; STATUS
            db    0                 ; ZULED: Led-Zustand
            dw    4000              ; RATE: Blink-Rate
            dw    1                 ; COUNTER

ENDP  X1P0380_I

;-----------------------------------------------------------------------------
;     Prozedur 2: LED ON
;-----------------------------------------------------------------------------
;     Vom Betriebssystem wird in dx bei Aufruf einer Prozedur immer die
;     Task-Nr uebergeben, das wird aber hier nicht ausgewertet.

PROC  X1P0380_2   FAR

            pusha
            push  ds
            push  es
            mov   ax,cs             ; Tiny-Model: ds auf Anfang setzen
            mov   ds,ax
            xor   ax,ax
            mov   es,ax
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            call  [DWORD es:LED_ON]
            mov   [BYTE ds:ZULED],1 ; neuen Zustand der LED speichern
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            pop   es
            pop   ds
            popa
            ret

ENDP  X1P0380_2
```

```
;-----------------------------------------------------------------------------
;     Prozedur 3: LED OFF
;-----------------------------------------------------------------------------
;     Vom Betriebssystem wird in dx bei Aufruf einer Prozedur immer die
;     Task-Nr uebergeben, das wird aber hier nicht ausgewertet.

PROC  X1P0380_3   FAR

            pusha
            push  ds
            push  es
            mov   ax,cs             ; Tiny-Model: ds auf Anfang setzen
            mov   ds,ax
            xor   ax,ax
            mov   es,ax
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            call  [DWORD es:LED_OFF]
            mov   [BYTE ds:ZULED],0 ; neuen Zustand der LED speichern
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            pop   es
            pop   ds
            popa
            ret

ENDP  X1P0380_3

;-----------------------------------------------------------------------------
-
      END X1P0380
;-------------------------- END OF X1P0380.ASM ------------------------------
```

Historie dieses Dokumentes

27.6.01	hb	1) Neuer Dateiname AN083_C.doc 2) alle Routinen für die Uhr (= RTC) komplett überarbeitet 3) Interrupt-Routinen um Int-Nr. 98h, 99h und 9ah für RTC erweitert 4) Routinen-Nr. 33, 34, 35, 45 und 46 angepaßt 5) GET_RTC_INT_STATUS (Nr. 118) neu 6) Subroutinen-Nr. 41 und 42 Name geändert in LOCAL_LED_...
26.6.01	hb	von hk übernommen: alter Dateiname = AN083_B.doc