PDV-E 119
Juni 1978

PDV-Entwicklungsnotizen

PEARL-Betriebssystem für den Z80

R. Rössler, Erlangen

Kernforschungszentrum Karlsruhe

PEARL - Betriebssystem

für den

Z80 - Rufbus-Subset

(Projekt 'PEARL im Nahverkehr')

Inhalt

1. Einleitung

Das Betriebssystem für den Rufbus-Subset wurde durch Auswahl von geeigneten Bausteinen aus einem modularen System aufgebaut, und bewältigt nur diejenigen Funktionen, die für den speziellen Anwendungsfall erforderlich sind. Dabei handelt es sich ausschließlich um einen eingeschränkten satz von "Tasking-" und Synchronisationsmechanismen, da beispielsweise keine Standard- Ein/Ausgabe vorgesehen ist. Es sind außerdem nur "Interrupt-Schedules" realisiert, da keine Zeit-Einplanungen benötigt werden.

Der Großteil des Betriebssystems liegt in einer anlagenunabhängigen Fassung vor (siehe Anhang I). Diese wird über Makros mit Hilfe des Makroprozessors STAGE2 in Z80-Assembler übersetzt. Der Rest, bei dem es sich hauptsächlich um Routinen zur Befriedigung der vorgegebenen Schnittstellen handelt, liegt in Assembler vor.

2. Anlagenunabhängiger Teil

2.1. Leistungsumfang

Der Funktionsumfang beschränkt sich auf Tasking- und Synchronisationsmechanismen und wird am besten in einer Notation beschrieben die an die PEARL-Syntax angelehnt ist.

2.1.1. Aktivieren

[WHEN interrupt] ACTIVATE task [PRIORITY integer];

Die betreffende Task wird bei Eintreffen des Interrupts in den Zustand "lauffähig" versetzt, sofern sie nicht bereits aktiv ist. Andernfalls ist der Aufruf wirkungslos. Aktivierungen werden nicht gepuffert.

Falls der Task durch die Prioritätsangabe (Ø...127) die größte Dringlichkeitsstufe unter den Konkurrenten um den Prozessor zugeteilt wird, so wird sie in den Zustand "laufend" versetzt. Das Betriebssystem erwartet in jedem Fall eine Prioritätsangabe. Falls diese auf Sprachebene fehlt, so muß vom Übersetzungssystem ein Ersatzwert vorgegeben werden (beispielsweise der in der Task-Deklaration angegebene Wert).

2.1.2. Beenden

TERMINATE [task];

Die angegebene (bzw. die aufrufende) Task wird in den Zustand

2.1.3. Anhalten

SUSPEND;

Die anrufende Task wird in den Zustand 'angehalten' versetzt.

2.1.4. Fortsetzen

[WHEN interrupt] CONTINUE [task];

Die betreffende Task wird bei Eintreffen des Interrupts in den Zustand "lauffähig" versetzt, sofern sie sich [dann] im Zustand "angehalten" befindet. Andernfalls ist der Aufruf wirkungslos. Ihre Priorität ist weiterhin die bei der Aktivierung angegebene. Falls die Taskangabe fehlt, d.h. falls sich die Anweisung auf die aufrufende Task bezieht, so muß eine Einplanungsbedingung angegeben sein.

2.1.5. Warten

WHEN interrupt RESUME;

Diese Anweisung entspricht der unteilbaren Folge

WHEN interrupt CONTINUE; SUSPEND;

2.1.6. Löschen von Einplanungen

PREVENT [task];

Aktivierungs- bzw. Fortsetzungseinplanungen für die betreffende Task werden vernichtet.

2.1.7. Semaphor-Operationen

REQUEST semaphore;
RELEASE semaphore;

Die Wirkung der Aufrufe entspricht der üblichen Konvention.

2.2. Verwendete Datenstrukturen

Die folgenden Datenstrukturen sind vorgesehen:

- Taskvewaltungsblöcke
- Semaphor-Verwaltungsblöcke
- Auftragsparameterblöcke
- Interruptliste
- Initialisierungsliste

Für die Verkettung von Verwaltungsdaten zur Laufzeit werden 2 Verfahren angewandt:

- Warteschlangen
- "Geschlossene" Ketten

2.2.1. Warteschlangen

Warteschlangen sind doppelt verzeigert (jedes Element enthält je einen Zeiger auf "Vorgänger" bzw. "Nachfolger"). Die Elemente sind nach Prioritätszahlen geordnet. Der Warteschlangenkopf enthält je einen Zeiger auf das erste bzw. letzte Element in der Warteschlange.

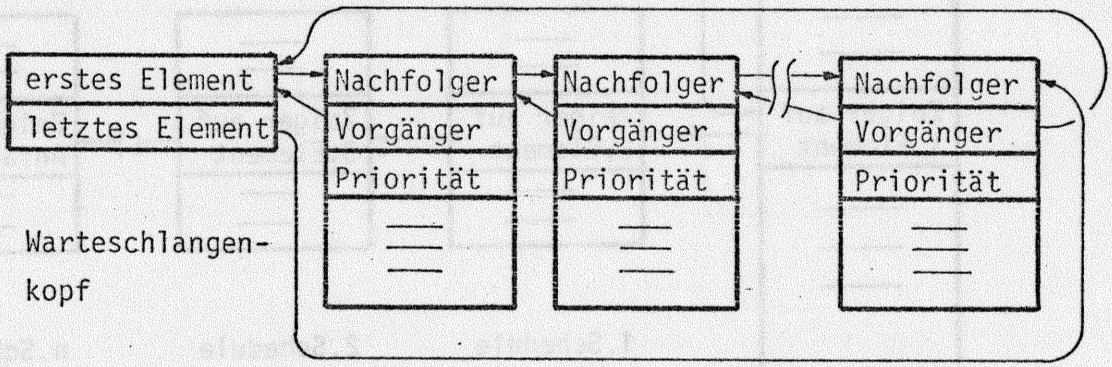


Bild 1: Prinzip der Verkettung von Taskverwaltungsblöcken über Warteschlangen

Es existieren die folgenden Warteschlangen:

- Prozessorwarteschlange (PWS) verkettet alle lauffähigen Tasks. Der Warteschlangenkopf liegt im Betriebssystem.
- Semaphor-Warteschlange (SWS) verkettet alle auf die Freigabe der betreffenden Semaphorvariablen wartenden Taks. Der Warteschlangenkopf ist Bestandteil des Semaphor-Verwaltungsblockes.

2.2.2. "Geschlossene" Ketten

Geschlossene Ketten sind nur einfach verzeigert, wobei das letzte Element der Kette auf den Kettenanfang zeigt. Sie werden zur Verzeigerung von Einplanungsbedingungen ("Schedules") verwendet, die dem selben Ereignis zugeordnet sind.

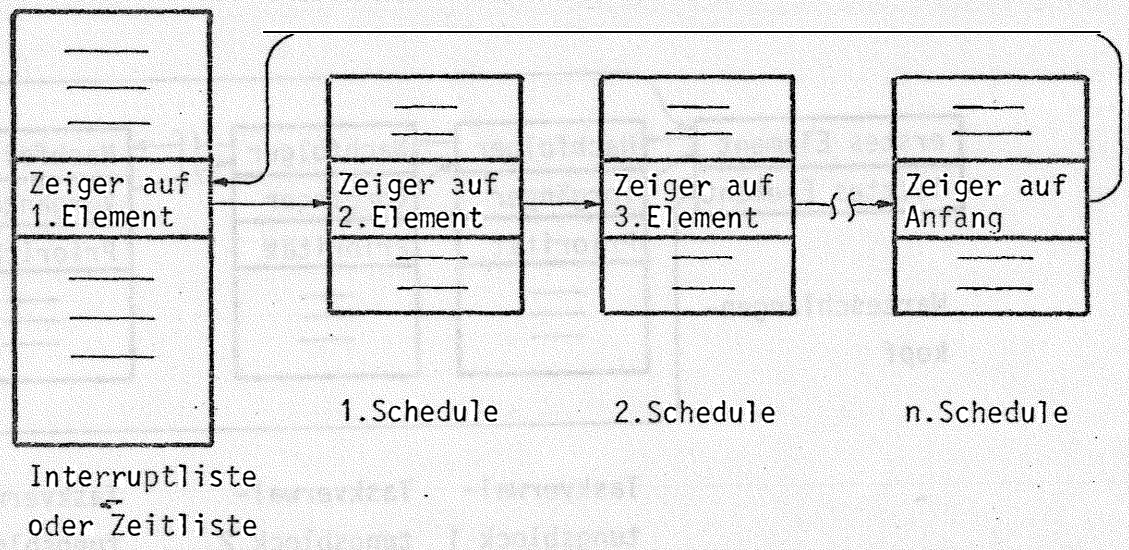


Bild 2: Schedule-Kette (geschlossene Kette)

Der Kettenanfang ist durch eine Zelle der Interruptliste repräsentiert und identifiziert das Ereignis, das zur Auswertung der betreffenden Einplanungsbedingungen führt. Die "geschlossene" Verkettung ist notwendig, da ein Ausketten eines Elementes möglich sein muß, ohne den Kettenanfang zu kennen. Die aus der einfachen Verzeigerung sich ergebende längere Bearbeitungszeit wurde hier in Kauf genommen und fällt dann nicht ins Gewicht, wenn die Wahrscheinlichkeit gering ist, daß sich mehrere Schedules auf das gleiche Ereignis beziehen.

2.2.3. Taskverwaltungsblock

In den Taskverwaltungsblöcken werden alle jene Informationen geführt, die den Zustand einer Task beschreiben (siehe Bild 3).

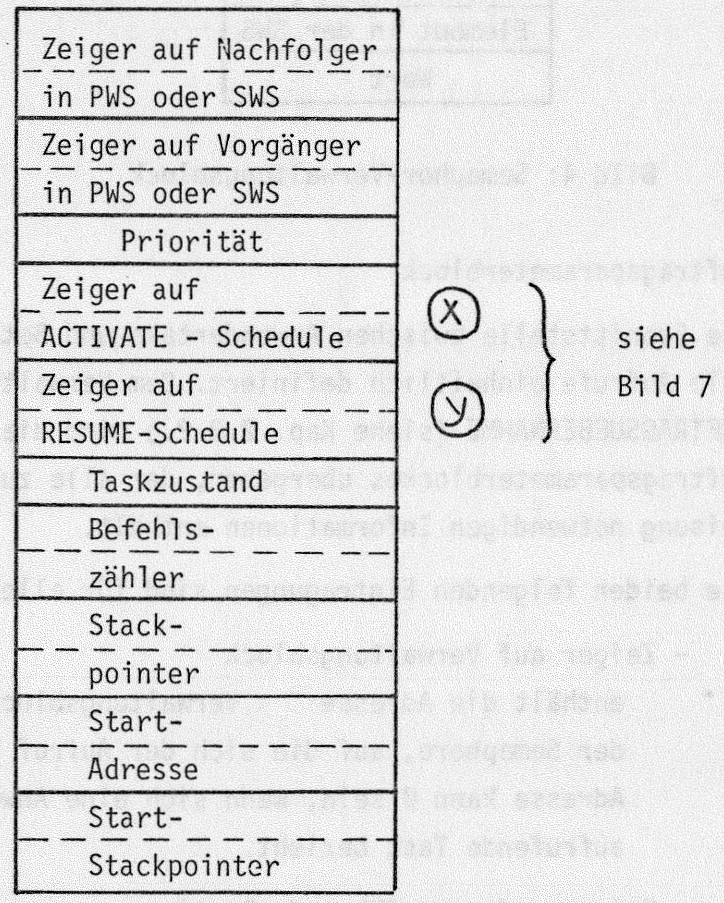


Bild 3: Taskverwaltungsblock

Der Speicherplatz für die Taskverwaltungsblöcke muß vom PEARL-Übersetzungssystem reserviert werden. Sie können an beliebigen Stellen im RAM zu liegen kommen. Die Vorbesetzungen werden von der Initialisierungsroutine vorgenommen (Kap. 2.3.1.)

2.2.4. Semaphor-Verwaltungsblock

Für jede Semaphor-Variable muß vom Übersetzer Speicherplatz für einen Verwaltungsblock reserviert werden. Die Einträge werden von der Initialisierungsroutine (siehe Kap. 2.3.1.) vorbesetzt, wobei der Wert jeder Semaphore auf Ø gesetzt wird (!).

Zeiger auf erstes Element in der SWS
Zeiger auf letztes Element in der SWS
Wert

Bild 4: Semaphor-Verwaltungsblock

2.2.5. Auftragsparameterblock

Die Schnittstelle zwischen Anwendertaks und Betriebssystem ist für alle Aufrufe einheitlich definiert. Dem Verwaltungsbaustein AUFTRAGSUEBERNAHME (siehe Kap. 2.3.2.) wird die Adresse eines Auftragsparameterblockes übergeben, der alle zur Ausführung der Anweisung notwendigen Informationen enthält.

Die beiden folgenden Eintragungen sind für alle Aufrufe äquivalent

- Zeiger auf Verwaltungsblock
  enthält die Adresse des Verwaltungsblockes der Task oder der Semaphore, auf die sich der Aufruf bezieht. Diese Adresse kann Ø sein, wenn sich eine Anweisung auf die aufrufende Task bezieht.

- Zeiger auf auszuführende Routine
  enthält die Adresse der für die Ausführung der betreffenden Anweisung zuständigen BS-Routine.

Die weiteren Informationen sind aufrufabhängig.

Bei einer Synchronisationsanweisung hat der Parameterblock das folgende Aussehen.

ohne Bedeutung
Zeiger auf Semaphor-Verwaltungsblock
Adr. der REQUEST- oder RELEASE-Routine
Ø

Bild 5: Auftragsparameterblock für REQUEST oder RELEASE

Für eine "Tasking"-Anweisung ohne Einplanungsbedingung sehen die Parameter wie folgt aus:

ohne Bedeutung
Zeiger auf Task-verwaltungsblock oder Ø
Zeiger auf zuständige BS-Routine
Ø
Priorität

nur bei 'ACTIVATE' erforderlich

Bild 6: Auftragsparameterblock für Task-Aufrufe ohne Schedule.

Die Auftragsparameterblöcke für einzuplanende Aufrufe enthalten noch zusätzliche Informationen

Schedule-Kettenzeiger	*)
Zeiger auf Task-verwaltungsblock oder Ø	
Zeiger auf zuständige BS-Routine	
12 (WHEN-Kennung)	
Priorität (ACTIVATE) oder Ø	
Interruptnummer	
aktuellerSchedule-Typ	*)
Zeiger auf (x) oder (y) (siehe Bild 3)	*)

Die mit *) gekennzeichneten Informationen werden vom Betriebssystem eingetragen. Sie können beliebig vorbesetzt sein.

Bild 7: Auftragsparameterblock für Einplanungen (Schedule)

2.2.6. Interruptliste

Jedem Prozeßalarm ist ein Listenplatz in der Interruptliste zugeordnet, der den Anfang einer möglichen Schedule-Kette darstellt. Falls für das betreffende Ereignis keine Aktion eingeplant ist, zeigt der zugehörige Zeiger auf sich selbst. Die Interruptliste ist Bestandteil des Betriebssystems - es braucht also vom Übersetzer kein Speicherplatz reserviert werden. Sie wird durch die Initialisierungsroutine vorbelegt.

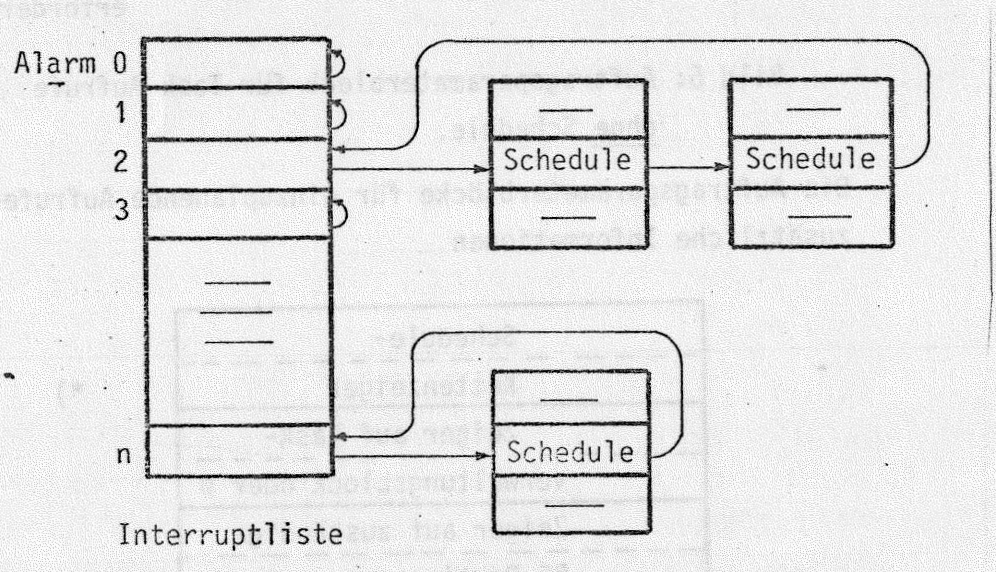


Bild 8: Interruptliste

2.2.7. Initialisierungsliste

Die Initialisierungsliste wird von der Initialisierungsroutine beim Neustart benötigt und ist vom Übersetzungssystem bereitzustellen. Sie enthält die folgenden Informationen:

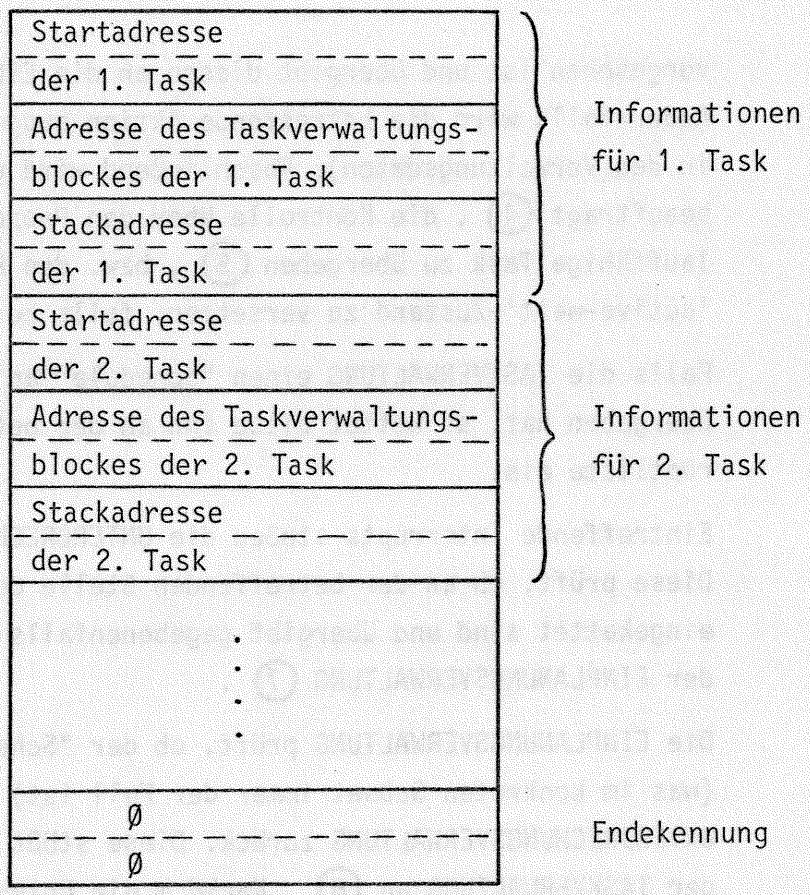


Bild 9: Initialisierungsliste

2.3. Verwaltungsbausteine

Das Betriebssystem enthält die folgenden 6 Verwaltungsbausteine

- Initialisierungsroutine
- Auftragsübernahme
- Unterbrechungsverwaltung
- Einplanungsverwaltung
- Taskverwaltung
- Prozessorverwaltung

Die Zusammenarbeit der Bausteine kann grob wie folgt beschrieben werden (siehe Bild 1Ø).

Alle Betriebssystemaufrufe, die von Tasks abgeben werden, werden von der AUFTRAGSUEBERNAHME entgegengenommen (1) . Diese initialisiert einen etwaigen "Schedule" und ruft den für den Auftrag zuständigen Modul der TASKVERWALTUNG auf (2). Dieser prüft, ob ein "Schedule"

vorgesehen ist und übergibt diesen an die EINPLANUNGSVERWALTUNG (3). Andernfalls wird die betreffende Aktion ausgeführt (über Manipulationen in den Verwaltungsdaten). Abschließend wird die PROZESSORVERWALTUNG beauftragt (4) , die Kontrolle über den Prozessor an die höchstpriore lauffähige Task zu übergeben (5) , bzw. den Prozessor in eine 'active-wait'-Zustand zu versetzen, falls keine solche existiert.

Falls die TASKVERWALTUNG einen "Schedule" an die EINPLANUNGSVERWALTUNG übergeben hat, so kettet diese ihn an der betreffenden Stelle der Interruptliste ein.

Eintreffende Interrupts stoßen die UNTERBRECHUNGSVERWALTUNG an (6) . Diese prüft, ob an der betreffenden Stelle der Interruptliste "Schedules" eingekettet sind und übergibt gegebenenfalls diese der Reihe nach der EINPLANUNGSVERWALTUNG (7) .

Die EINPLANUNGSVERWALTUNG prüft, ob der "Schedule" bereits abgelaufen ist, (was im konkreten Subset immer der Fall ist) und meldet dies der UNTERBRECHUNGSVERWALTUNG zurück. Diese stößt den zuständigen Modul der TASKVERWALTUNG an (8) . Nachdem die Unterbrechungsverwaltung alle in Frage kommenden Schedules übergeben hat, wird abschließend wieder die PROZESSORVERWALTUNG angestoßen (9) .

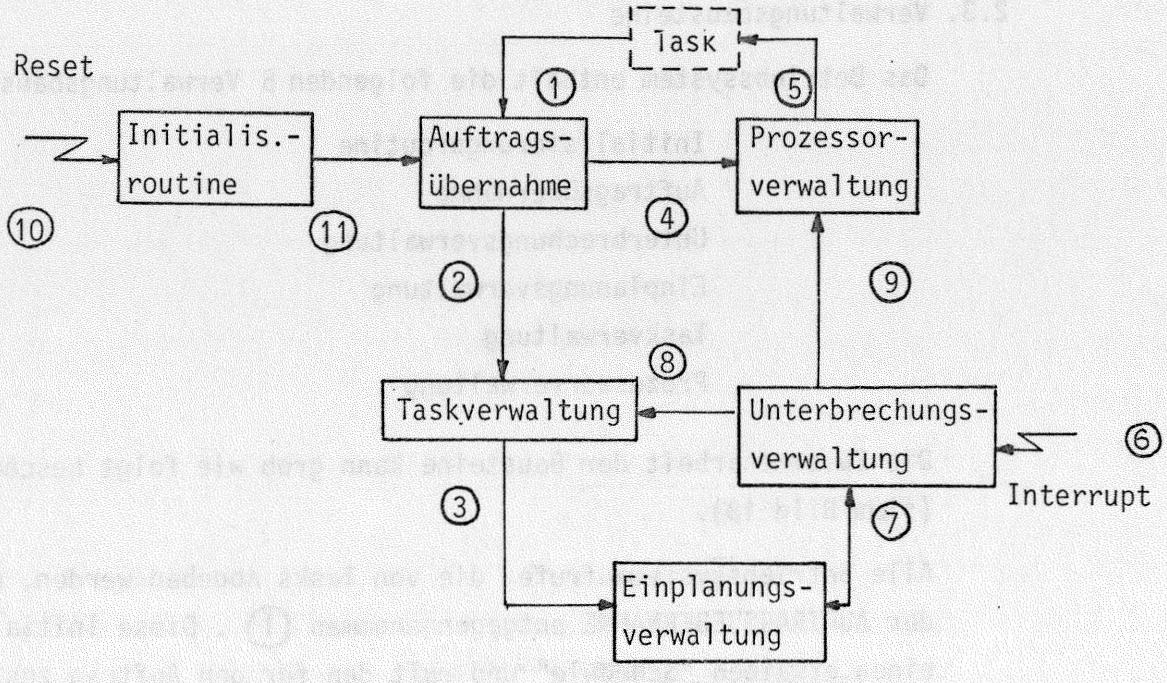


Bild 10: Zusammenarbeit der Verwaltungsbausteine

Die INITIALISIERUNGSROUTINE wird bei "Reset" angestoßen (10) . Sie führt anlagenabhängige Aktionen aus, obwohl sie in anlagenunabhängiger Repräsentation vorliegt:

Sie hat die folgenden Aufgaben:

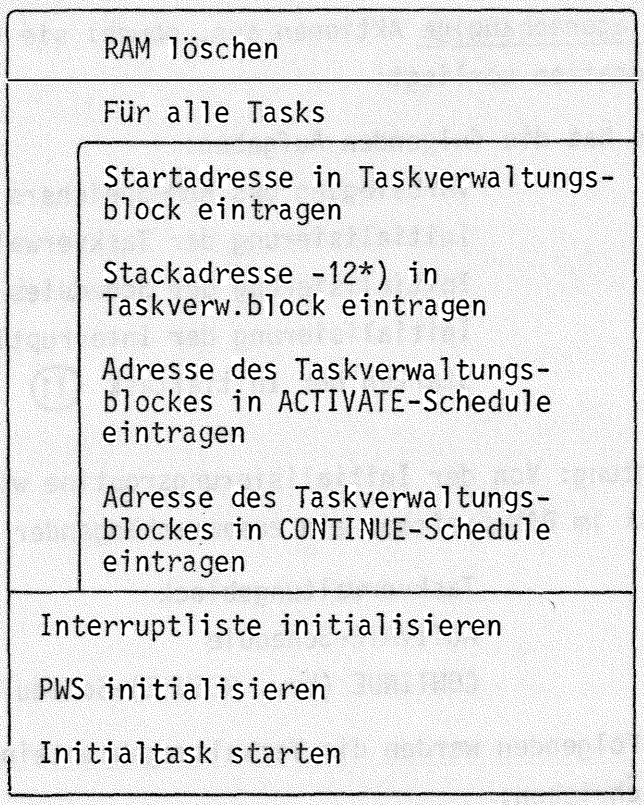
- Vorbelegung des RAM-Speichers mit Ø
- Initialisierung der Taskverwaltungsblöcke
- Initialisierung der Schedules
- Initialisierung der Interruptliste
- Starten der Initialtask (11)

Achtung: Von der Initialisierungsroutine wird vorausgesetzt, daß je Task im RAM-Speicher ein zusammenhängender Bereich reserviert ist für

- Taskverwaltungsblock
- ACTIVATE-Schedule
- CONTINUE (bzw. RESUME)-Schedule

Im Folgenden werden die Verwaltungsbausteine des Betriebssystems im Detail beschrieben:

2.3.1. Initialisierungsroutine (Marke 'INIT')



*) Im Stack wird Platz zum Retten von 6 Registern geschaffen.

Bild 11: Initialisierungsroutine

2.3.2. Auftragsübernahme (Marke 'SVC')

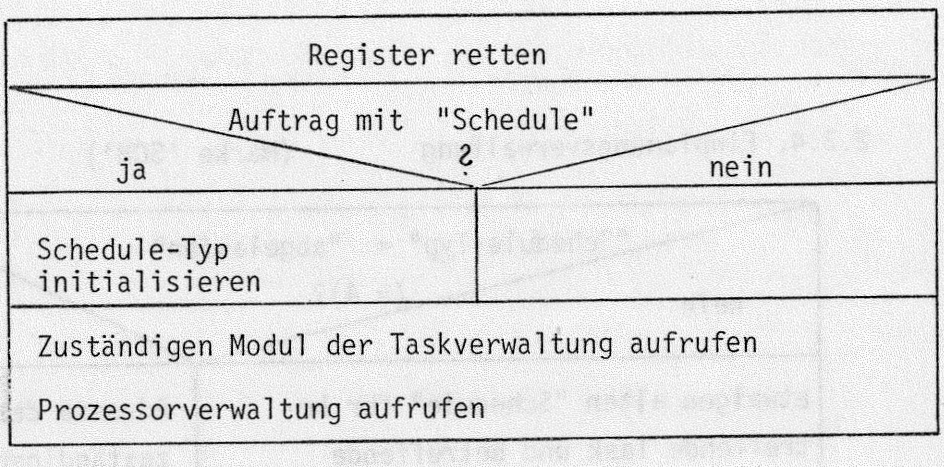


Bild 12: Auftragsübernahme

2.3.3. Unterbrechungsverwaltung (Marke 'ITR')

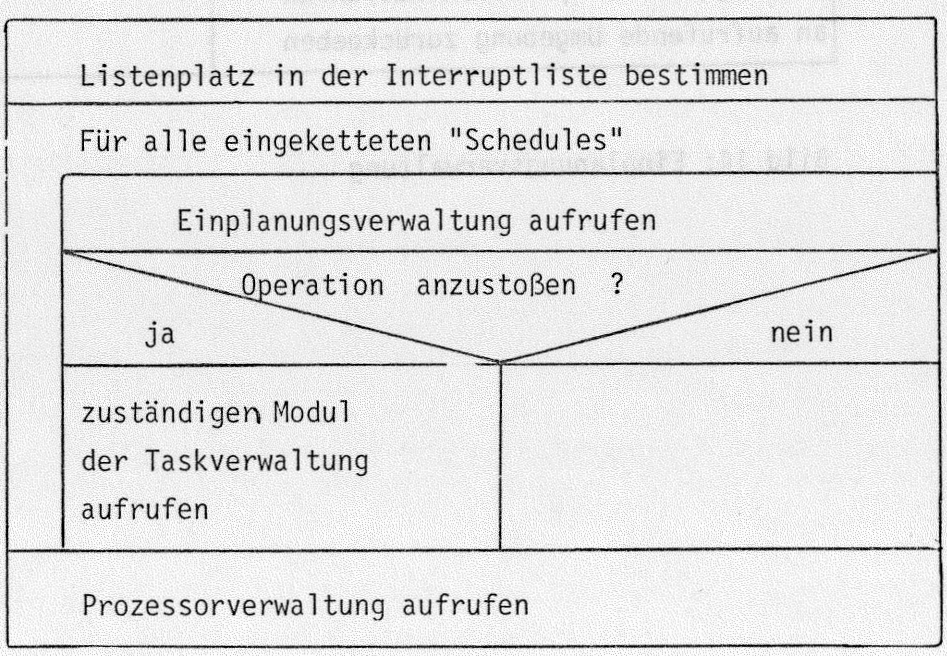


Bild 13: Unterbrechungsverwaltung

2.3.4. Einplanungsverwaltung (Marke 'SCH')

"Schedule-Typ" = "abgelaufen" (= 4)? nein	ja
etwaigen alten "Schedule" für betreffende Task und betreffende Operation ausketten In Taskverwaltungsblock Adresse des neuen "Schedules" eintragen "Schedule-Typ" (12 = WHEN) um 8 erniedrigen (→4 = "abgelaufen") Über Interruptnummer "Schedule" in Interruptliste einketten Kennung 'keine Operation ausführen' an aufrufende Umgebung zurückgeben	Adresse der zuständigen Routine an aufrufende Umgebung zurückgeben

Bild 14: Einplanungsverwaltung

2.3.5. Taskverwaltung

Die Taskverwaltung enthält die Moduln

ACTIVATE
TERMINATE
SUSPEND
CONTINUE
RESUME
REQUEST
RELEASE
PREVENT

2.3.5.1. Der Modul ACTIVATE (Marke 'START')

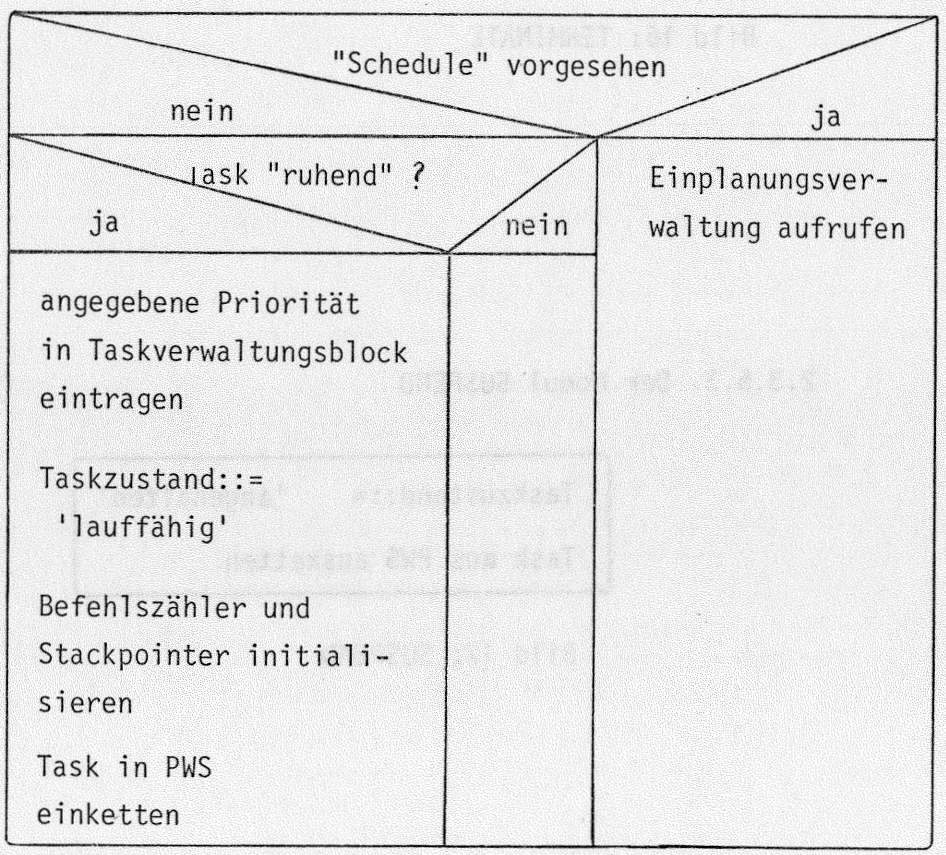


Bild 15: ACTIVATE

2.3.5.2. Der Modul TERMINATE (Marke 'STOP')

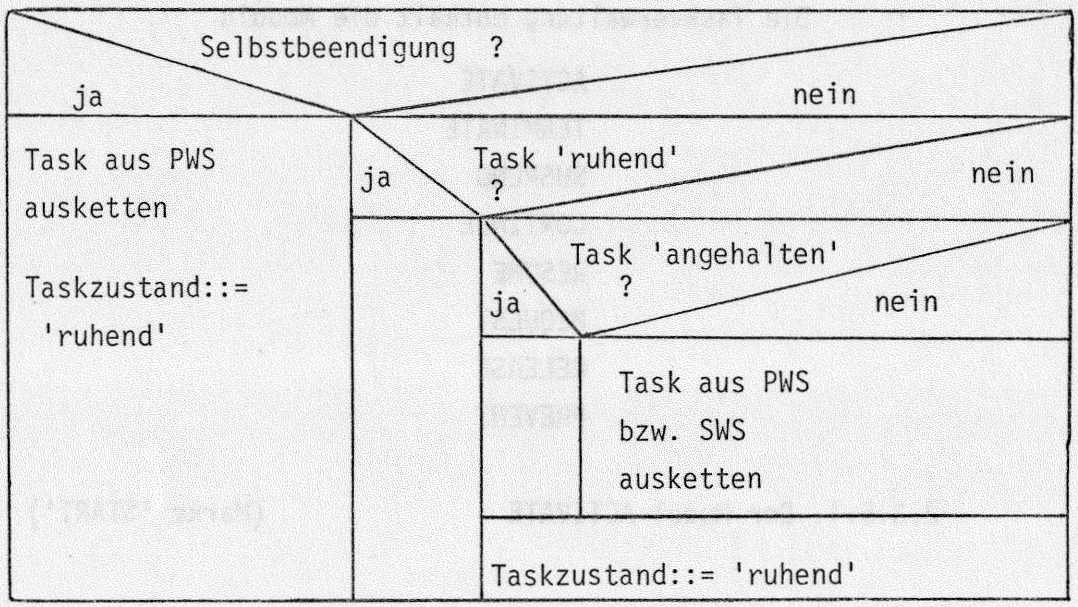


Bild 16: TERMINATE

2.3.5.3. Der Modul SUSPEND (Marke 'SUSP')

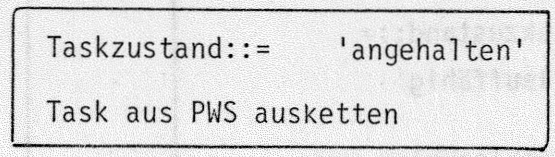


Bild 17: SUSPEND

2.3.5.4. Der Modul CONTINUE (Marke 'CONT')

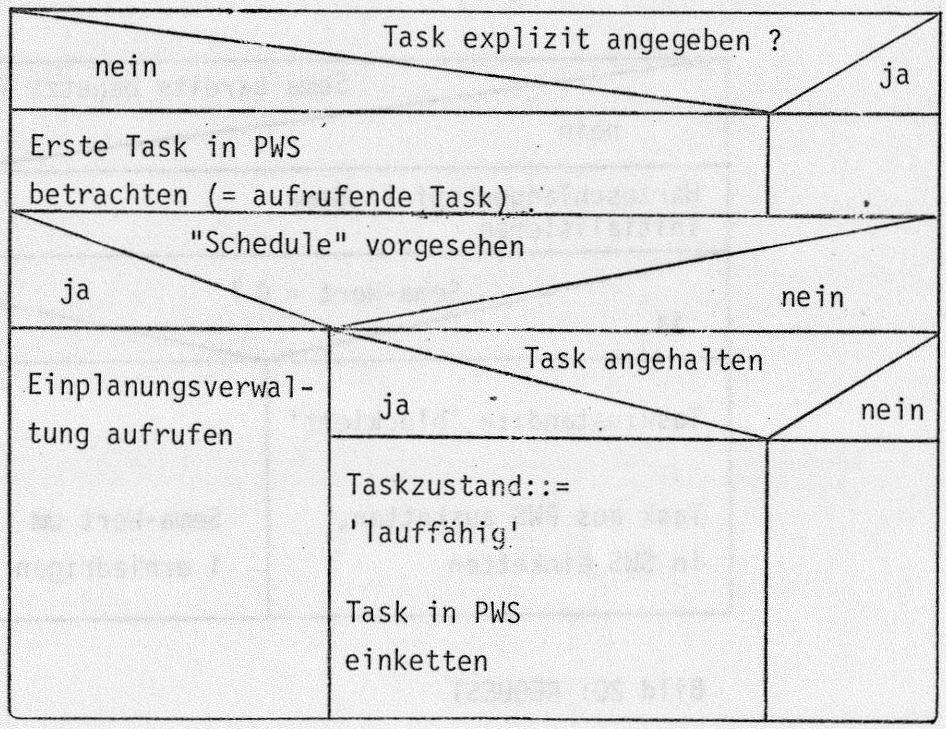


Bild 18: CONTINUE

2.3.5.5. Der Modul RESUME (Marke 'RESU')

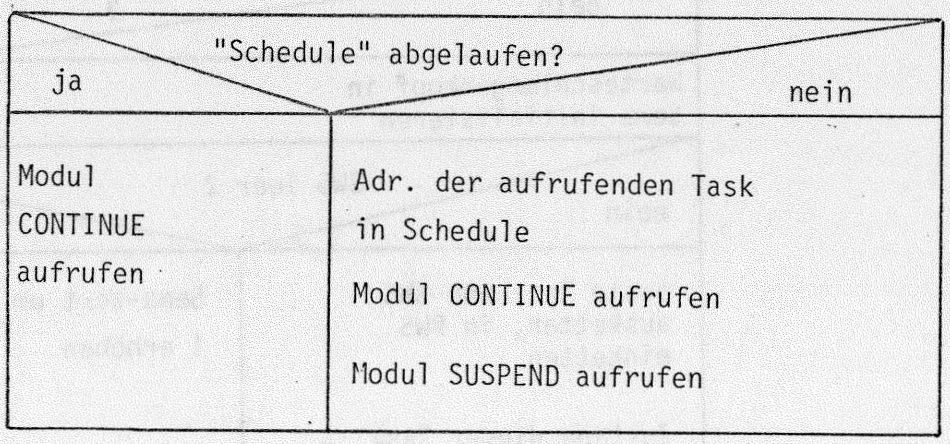


Bild 19: RESUME

2.3.5.6. Der Modul REQUEST (Marke 'REQU')

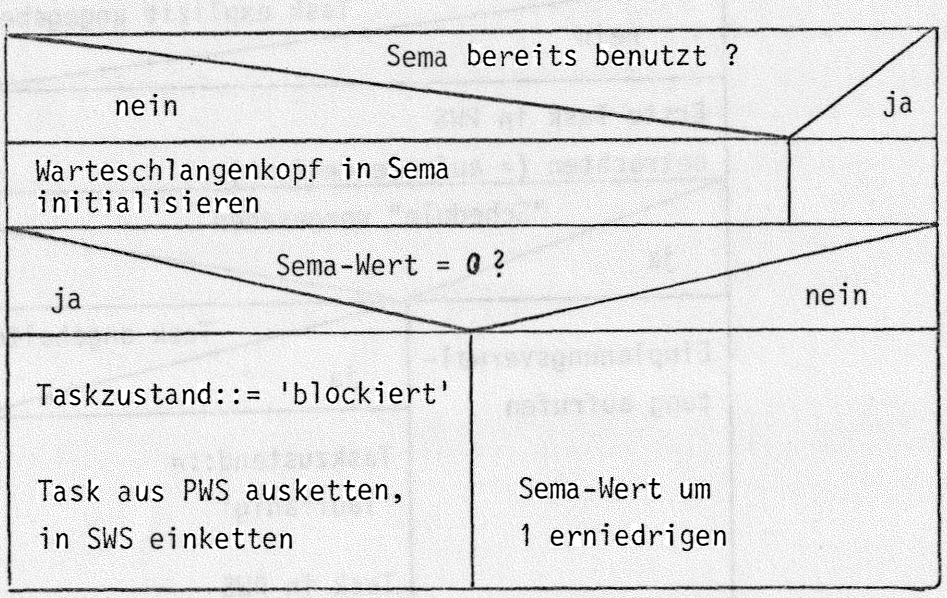


Bild 20: REQUEST

2.3.5.7. Der Modul RELEASE (Marke 'RELE')

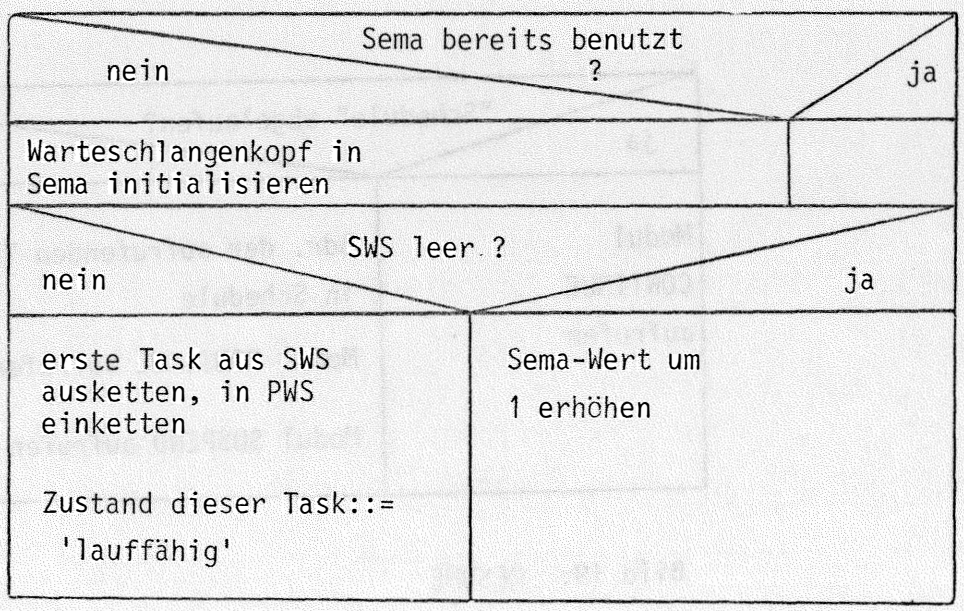


Bild 21: RELEASE

2.3.5.8. Der Modul PREVENT (Marke 'PREV')

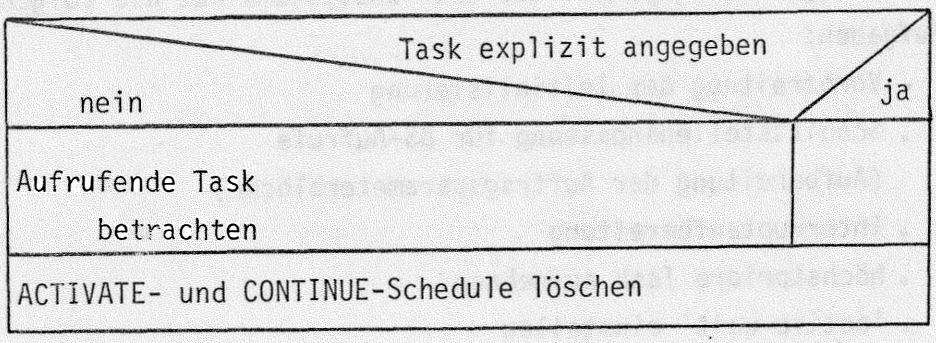


Bild 22: PREVENT

2.3.6. Prozessorverwaltung (Marke 'ASSIGN')

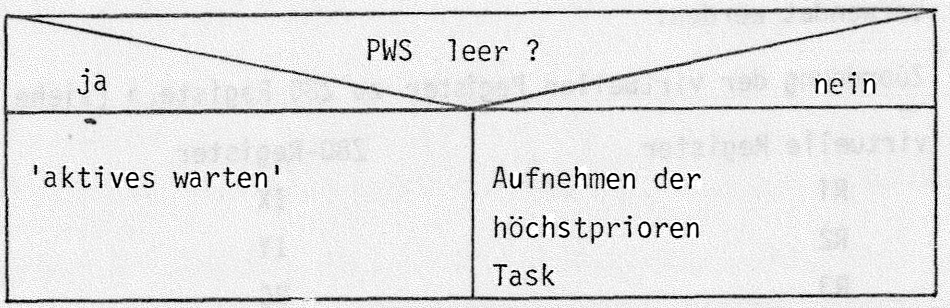


Bild 23: Prozessorverwaltung

3. Anlagenabhängiger Teil

3.1. Funktionsumfang

Der anlagenabhängige Teil des Betriebssystems hat die folgenden Aufgaben:

- Vorbereitung der Initialisierung
- Schnittstellenanpassung für BS-Aufrufe
  (Aufbereitung der Auftragsparameterblöcke)
- Interruptaufbereitung
- höchstpriore Task aufnehmen
- 'active-wait' einstellen
- Simulation von BC als Basisregister

3.2. Registerkonventionen

Die Register A' und HL' des zweiten Registersatzes sind für die Systemzustandskennung und die Zwischenspeicherung von Interrupts reserviert. Der zweite Registersatz darf nur unter Interruptsperre verwendet werden.

Zuordnung der virtuellen Register zu Z80 Registern (siehe Anhang I)

virtuelle Register	Z80-Register
R1	IX
R2	IY
R3	BC
R4 ÷ R7	Speicherzellen R4÷R7

3.3. Systemzustände

Das System kann sich in 2 Zuständen befinden:

Zustand 'Betriebssystem tätig': A' = Ø
Zustand 'Anwenderebene' : A' = 1

Im Zustand 'Betriebssystem tätig' werden eintreffende Interrupts nicht sofort verarbeitet, sondern zwischengespeichert. Ihre Verarbeitung erfolgt am Ende der laufenden BS-Aktion.

3.4. Beschreibung der Routinen

3.4.1. Vorbereitung der Initialisierung

(Marke 'ANFANG')

Dieser Baustein wird bei 'RESET' angesprungen.
Er erfüllt die folgenden Funktionen:

- Laden des Stackpointers (BS-interner Stack)
- I-Register laden, Interruptmodus 2 einschalten
- Zustand 'BS tätig' einschalten
- Interruptkeller initialisieren
- IY mit Adresse des 1. Taskverwaltungsblockes laden
- Initialisierungsroutine anstoßen (2.3.1.)

3.4.2. Schnittstellenanpassung

Der Baustein sorgt für die Rettung der Arbeitsregister in den taskspezifischen Keller sowie für die Aufbereitung des Auftragsparameterblockes gemäß Kap. 2.2.5. und dessen Übergabe an die Auftragsübernahme (Kap. 2.3.2.).

3.4.2.1. Aufträge ohne Schedule

Die Aufrufsequenz für Aufträge ohne Schedule sieht wie folgt aus:

```
CALL  BS
DW    verw
DW    oper
DB    Ø
DB    prio
```

verw.... Adresse der betreffenden Verwaltungsstruktur (Taskverwaltungsblock, Sema).
Ø = aufrufende Task

oper... Marke des zuständigen Moduls der Taskverwaltung (Kap.2.3.5.)

prio.... Priorität bei ACTIVATE, sonst Ø

Da die Daten durch diese Anordnung bereits in der richtigen Reihenfolge ein Speicher stehen, stellen sie unmittelbar den Auftragsparameterblock dar.

Die Schnittstellenanpassung führt hier die folgenden Aktionen durch (Marke 'BS'):

- Kellern der Arbeitsregister
- Adresse der Parameter nach IX
- Zustand 'BS tägig' einschalten
- Rücksprungadresse und Stackpointer in Taskkontrollblock retten
- Stackpointer auf BS-Stack stellen
- Auftragsverwaltung anstoßen (Kap. 2.3.2.)

3.4.2.2. Aufträge mit Schedule

Die Aufrufsequenz für Aufträge mit Schedule sieht wie folgt aus:

```
CALL  SCHED
DW    tcb + kenn
DW    oper
DB    12D
DB    prio
DB    itrn
```

tcb..... Adresse des Taskverwaltungsblockes
(Ø = aufrufende Task)

kenn.... 18D bei ACTIVATE
30D bei CONTINUE / RESUME

oper.... Marke des zuständigen Moduls der Taskverwaltung (Kap.2.3.5.)

prio.... Priorität bei ACTIVATE, sonst Ø

itrn.... Interruptnummer (1...19)

Bei Aufträgen mit Schedule werden im Auftragsparameterblock durch das BS Einträge vorgenommen. Er muß daher im RAM liegen. Dies wird dadurch erreicht, daß die Schnittstellenanpassung die Parameter in den für

Schedules vorgesehenen Bereich hinter dem betreffenden Taskverwaltungsblock überträgt.

Folgende Aktionen werden ausgeführt (Marke 'SCHED'):

- Kellern der Arbeitsregister
- Adresse des Schedules bestimmen → IX
- Zustand 'BS tätig' einschalten
- Parameter in Schedule umspeichern
- Rücksprungadresse und Stackpointer in Taskverwaltungsblock retten
- Stackpointer auf BS-Stack stellen
- Auftragsverwaltung anstoßen (Kap. 2.3.2.)

3.4.3. Interruptaufbereitung

Für jeden Interrupt ist eine kurze Service-Routine bereitzustellen, die die folgenden Aktionen ausführt:

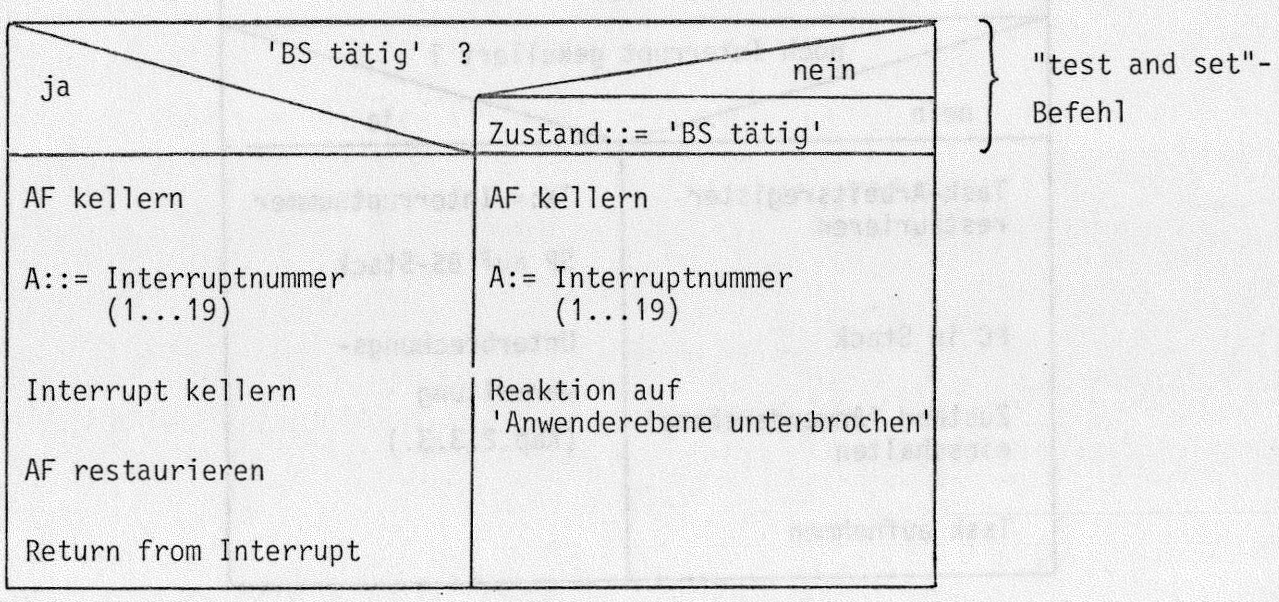


Bild 24: Service-Routinen für Interrupts

Für alle Interrupts gemeinsem ist die Reaktion auf 'Anwenderebene unterbrochen' (Marke 'ANW'):

- Interruptnummer retten (→ INR)
- HL und AF in Task-Stack
- PC retten (→ Variable PC)
- Niedrigpriore Interrupts wieder zulassen
- restliche Arbeitsregister kellern
- Stackpointer und PC in Taskverwaltungsblock retten
- Interruptnummer nach IX
- Stackpointer auf BS-Stack
- Unterbrechungsverwaltung anstoßen (Kap. 2.3.3.)

3.4.4. Aufnehmen der höchstprioren Task

Dieser Baustein wird von der Prozessorverwaltung (Kap. 2.3.6.) angesprungen. Er erfüllt die folgenden Funktionen:

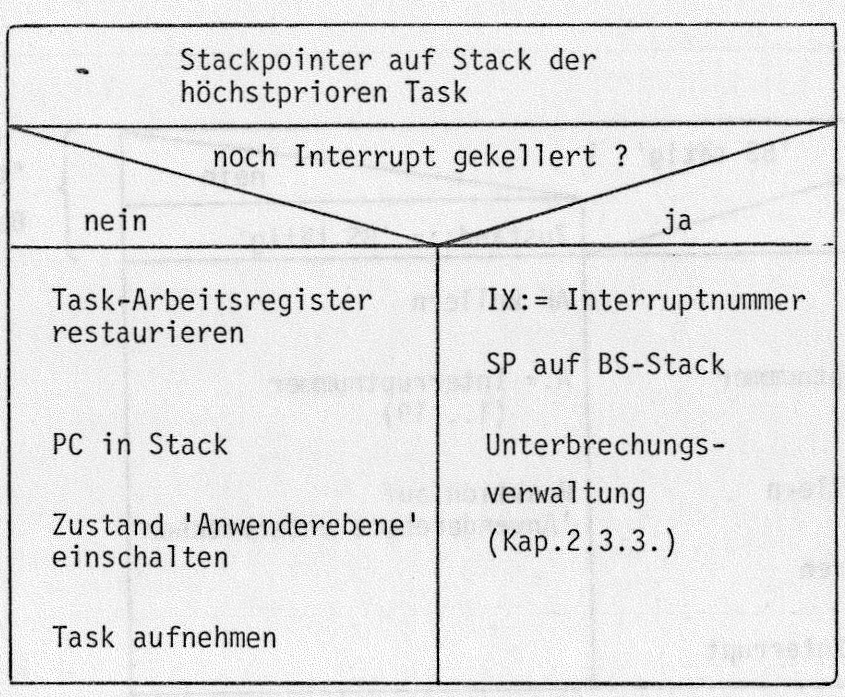


Bild 25: Aufnehmen höchstpriore Task

3.4.5. Prozessor in Zustand 'active-wait' versetzen (Marke 'ENABL')

Dieser Baustein wird von der Prozessorverwaltung (Kap. 2.3.6.) angesprungen, wenn sich keine Task im Zustand 'lauffähig' befindet.

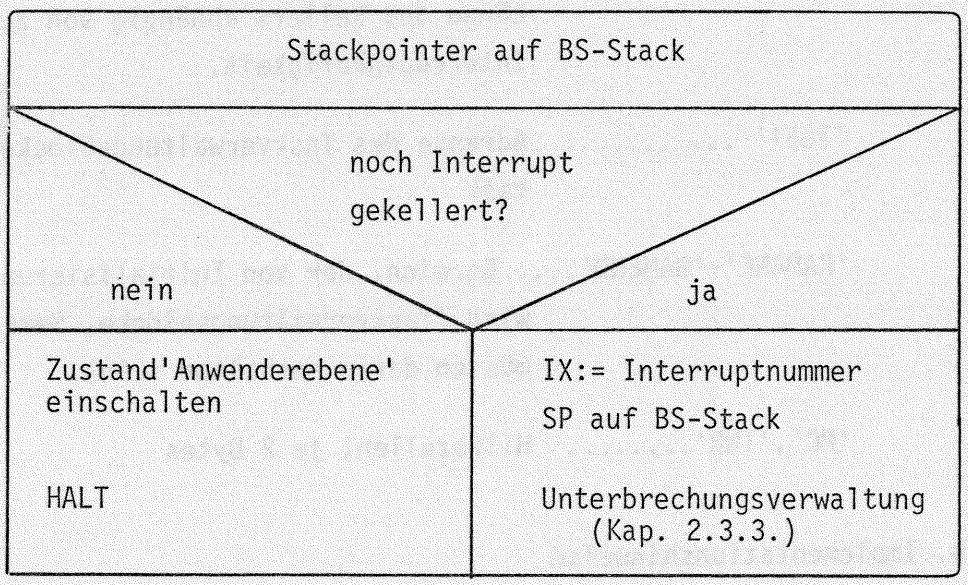


Bild 26: Prozessor in Zustand 'active-wait' versetzen

3.4.6. Simulation von BC als Basisregister

Es existieren 3 Routinen

R3L2	HL= (BC + A)
R3S2	(BC + E) := HL
R3L1	A:= (BC + A)

3.5. Daten und Speicherbereichsvereinbarungen

Der vom Übersetzungssystem berechnete Speicherbedarf für die taskspezifischen Keller muß um etwa 30 Bytes je Task erhöht werden. Dieser Platz wird vom Betriebssystem bei Unterbrechungen benötigt.

'ORGST'..... Adresse der letzten BS-Stack-Zelle + 1.
Länge des Tacks ca. 50 Bytes.

'ISPEI' ..... Adresse der ersten Zelle des Interruptkellers.
Länge des Kellers abhängig von zu erwartender
Interrupthäufigkeit.

'TCB1' ..... Adresse des Taskverwaltungsblockes der Initial-
task

'RAMANF'-'RAMEND'.... Bereich, der von Initialisierungsroutine gelöscht
wird (Taskverwaltungsblöcke, Semas und Schedules
müssen darin enthalten sein).

'PC','INR'..... Hilfszellen, je 2 Bytes

4. Implementationshinweise

Der in der Sprache 'SPASS' vorliegende anlagenunabhängige Teil des Betriebssystems wird mit dem Makroprozessor STAGE2 über die vorliegenden Makros in Z80-Assembler übersetzt. Das entstehende File ist zusammen mit dem anlagenabhängigen Teil in Maschinencode zu übersetzen. Es ist zweckmäßig, die von den Anwendertasks benötigten Adressen im Betriebssystem bei der Übersetzung des PEARL-Moduls über 'EQU'-Anweisungen bereitzustellen, da bei einer gemeinsamen Übersetzung von Betriebssystem und PEARL-Modul Mehrfachdefinitionen nicht auszuschließen sind.

Anhang I

BESCHREIBUNG DER PROGRAMMIERSPRACHE 'SPASS'

Beschreibung der Programmiersprache 'SPASS'
(System-Programmiersprache auf Assemblerniveau)

1. Einleitung

Die Sprache SPASS wurde speziell für die Implementation eines Modellbetriebssystems entwickelt, das zur experimentellen Untersuchung von Betriebsmittelzuteilungsstrategien für kleine Prozeßrechner dienen soll. Ihre Eigenschaften wurden so gewählt, daß eine einfache Übersetzung in Zielmaschinen-Assemblercode erfolgen kann (z.B. über den Makroprozessor STAGE-2), daß zudem beim Einsatz moderner Maschinen (z.B. PDP-11, SIEMENS 330) effektiver Code entsteht, und daß trotzdem die Erstellung möglichst dokumentations- und wartungsfreundlicher Programme ermöglicht wird. Dies wird vor allem durch die weitgehende Formatfreiheit, durch die sich an höhere Sprachen anlehnende Notation, und durch beliebig schachtelbare Strukturdefinitionen erreicht.

2. Die virtuelle Maschine

Die Sprache SPASS basiert auf einer virtuellen Maschine, die mit einem Speicher und 7 Registern arbeitet. Konstante, Register und Speicherzellen können nach bestimmten Vorschriften durch Operatoren verknüpft werden. Es sind keine Rechenregister definiert, sodaß eine Abbildung auf Ein- und Mehradreßmaschinen erleichtert wird.

2.1. Operanden

a. Konstante werden als Direktoperanden betrachtet

b. Die Register 1-7 und Speicherzellen können unmittelbar über ihren Namen (ihre symbolische Adresse) angesprochen werden.

c. Speicherzellen können indirekt über die Register 1-3 - bei zusätzlicher Angabe eines festen Adreßanteils (Offset) - adressiert werden.

d. Speicherzellen und Register können die Adresse des Operanden enthalten (Substitution).

2.2. Datentypen

a. INTEGER

b. POINTER

c. BITKETTE

Die Datentypen INTEGER und POINTER können in arithmetischen Ausdrücken vorkommen.
Daten vom Typ BITKETTE können verglichen werden und über eine Bitselektion als Entscheidung für bedingte Sprünge verwendet werden.
Für alle 3 Datentypen sind Zuweisungen erlaubt.
Durch sog. Längendefinitionen wird angegeben, wieviele Speichereinheiten durch den jeweiligen Datentyp belegt werden. Eine einfache Übersetzung in Zielmaschinencode ist nur dann möglich, wenn alle 3 Längen gleich sind.

2.3. Befehle

2.3.1. Verknüpfungsoperatoren

Die folgende Tabelle zeigt, in welcher Form die erwähnten Datentypen verknüpft werden dürfen.

Operator	1.Operand	2.Operand	Ergebnis
Addition Subtraktion	Integer	Integer	Integer
	Pointer	Pointer (konstante)	Pointer
Multiplikation	Integer	Integer	Integer
	Pointer*	Integer	Pointer
Vergleich: (EQ,NE,GT,LT LE,GE)	Integer	Integer	TRUE|FALSE**
	Pointer	POINTER	TRUE|FALSE**
EQ,NE	Bitkette	Bitkette	TRUE|FALSE**
Selektion	Bitkette	Integer***	TRUE | FALSE**

* Es existiert eine (anlagenabhängige) Adresskonstante '$POINTER', die die Adreßdifferenz zweier aufeinanderfolgender POINTER-Größen darstellt (Repräsentation der Länge einer POINTER-Größe in Adreßeinheiten). Durch Multiplikation dieser Konstanten mit Integer-Größen werden die Adressen von POINTER-Feldelementen bestimmt.

** Die Ergebnisse von Vergleichsoperationen und der Bitselektion stellen Wahrheitswerte dar und werden nur für bedingte Sprünge verwendet.

*** Bei einer Bitselektion tritt als zweiter Operand immer nur eine Integer-Konstante auf, deren Maximalwert 8 ist.

2.3.2. Zuweisung

Ergebnisse von Ausdrücken können Registern oder Speicherzellen zugewiesen werden, wobei alle unter 2.1 (b-d) erwähnten Adressierungsarten möglich sind.

2.3.3. Sprungbefehle

a. Unbedingter Sprung (direkt oder indirekt ohne Offset)
b. Bedingte Sprünge (- " -)
c. Unterprogrammsprung(- " -)
d. Rücksprung aus Unterprogramm

2.3.4. Organisatorische Befehle

a. Retten der Register eines Rechenprozesses
b. Register restaurieren und ausgewählten Rechenprozeß aufnehmen
c. Interrupt sperren
d. Interrupt freigeben
e. Geräteversorgung
f. Fehlerstop (bei nicht behebbaren Fehlersituationen)
g. Warten auf Interrupt

3. Detaillierte Beschreibung der Sprachelemente

Im folgenden werden die Elemente von SPASS in einer modifizierten BNF-Notation angegeben, sowie in ihrer semantischen Bedeutung beschrieben. Nichtterminale Symbole werden durch Kleinschreibung repräsentiert. Die folgendne Begriffe werden nicht mehr weiter produziert:

name: Ein Name, der den üblichen Regeln genügt.
zahl: Eine Folge von Dezimalziffern
binärzahl: Eine Folge von Binärziffern (0 | 1)
kommentar: Folge beliebiger abdruckbarer Zeichen

Die Buchstabenfolge EOL repräsentiert ein 'Zeilenendezeichen' bzw. das Ende einer Lochkarte. Zwischenräume können hinter dem ersten Zeichen einer Zeile an beliebiger Stelle eingestreut werden.

3.1. Elementartyp

Ein 'Elementartyp' ist einer der 3 Typen 'INTEGER', 'POINTER', 'BITKETTE'.

elementartyp::= INT | Pointer | BITS

3.2. Typ

Ein 'Typ' ist entweder ein Elementartyp, oder eine Struktur, die durch eine vorhergegangene Strukturvereinbarung eingeführt wurde.

typ::= elementartyp | strukturtyp
strukturtyp::= name

3.3. Längenvereinbarung

Durch eine Längenvereinbarung wird der (anlagenabhängige) Speicherbedarf der Elementartypen in adressierbaren Einheiten definiert.

längenvereinbarung::=
␣LENGTH: elementartyp = zahl; eol

eol::= kommentar EOL
[{/kommentar; eol}...]

Beispiel:

␣LENGTH: POINTER = 2;

3.4. Strukturvereinbarung

Durch eine Strukturvereinbarung wird ein neuer Typ (Strukturtyp) definiert. Die Typen der Strukturelemente müssen entweder elementar sein oder bereits eingeführte Strukturtypen.

Der 'strukturtyp' in der ersten Zeile ist ein Name, durch den im folgenden der Typ der Struktur identifiziert wird. Durch die Elementzeilen werden die Strukturelemente beschrieben. Der 'name' in einer Elementzeile definiert den konstanten Offset (bezogen auf den Anfang der Struktur), über den auf das Strukturelement zugegriffen werden kann.

Der 'typ' in einer Elementzeile ist der Typ des Elementes.
Die Endezeile enthält wieder den in der ersten Zeile definierten Namen.

strukturvereinbarung::=
  ␣STRUCT : strukturtyp; eol
[{␣ELEM   : offset = typ; eol}...]
  ␣STREND : strukturtyp; eol
  offset::= name

Beispiel:

```
   STRUCT : KETTE;
     ELEM  : VORIGER = POINTER;
     ELEM  : NAECHSTER = POINTER;
   STREND : KETTE;
```

'KETTE' ist ein Strukturtyp, der aus 2 POINTER-Typen besteht. Der konstante Offset für den Zugriff auf die Elemente ist durch 'VORIGER' bzw. 'NAECHSTER' definiert.

3.5. Konstantennotationen

Konstante::= integerkonstante | adreßkonstante | bitkettenkonstante

3.5.1. Integerkonstante

'Kzahl' repräsentiert eine positive und
'KMzahl' eine negative Integerkonstante

integerkonstante::=
        'Kzahl' | 'KMzahl'

3.5.2. Adreßkonstante

Eine Adreßkonstante ist entweder eine Speicheradresse, ein konstanter Offset oder der Zeiger 'auf nichts'.

adreßkonstante::=
    $adresse | $offset | $NIL
adresse::= name
Hinweis: Die (anlagenabhängige) Adreßkonstante $POINTER stellt die Adreßdifferenz zweier aufeinanderfolgender POINTER-Variablen dar.

3.5.3. Bitkettenkonstante

Eine Bitkettenkonstante besteht aus bis zu 8 Binärziffern. Sie ist rechts mit Nullen aufgefüllt zu denken, bis die (anlagenabhängige) Darstellung einer Bitkette erreicht ist.

bitkettenkonstante::=
'B binärzahl'

3.6. Speicherreservierung

Mit einer Speicherreservierungsanweisung wird Platz für ein Element (oder ein eindimensionales Feld von Elementen) eines beliebigen Typs reserviert. Gleichzeitig kann eine Initialisierung mit Konstanten erfolgen.

speicherreservierung::=

```
␣SPACE : adresse =[zahl*] typ [+] ; eol
[[{␣[zahl*] SPEL   = konstante +; eol}...]
  ␣[zahl*] SPEL   = konstante; eol ]
```

Beispiel:

```
␣SPACE :  VEKTOR = 10 * INT +;
␣5 * SPEL = 'K5' +;
␣ SPEL = 'KM1';
```

Es wird ein Integer-Feld der Länge 10 mit dem Namen 'VEKTOR' definiert. Die ersten 5 Elemente werden mit 5 und das sechste wird mit -1 vorbesetzt.

3.7. Markenvereinbarung

markenvereinbarung::=

```
␣LOC : adresse; eol
```

3.8. Registernotation

register::= universalregister | R4 | R5 | R6 | R7
universalregister::= R1 | R2 | R3

3.9. Zugriff auf Speicherzellen

Der Zugriff auf Speicherzellen kann entweder direkt über eine Adresse,indiziert (Adresse = Registerinhalt + Offset) oder substituiert (Adresse = Registerinhalt oder Speicherzelleninhalt) erfolgen.

speicherzelle::=
adresse | inidzierung | substitution

indizierung::= offset, universalregister

substitution::= $Ø, adresse | $Ø, register

Beispiele:

```
ADAM  DISTANZ,R2  $Ø, ADAM   $Ø,R7
```

3.10. Operanden

Ein Operand kann eine Konstante, ein Register oder eine Speicherzelle sein.

operand::= konstante | register | speicherzelle

3.11. Ausdrücke

Es wird zwischen arithmetischen Ausdrücken und Bedingungen unterschieden.
Arithmetische Ausdrücke können auf der rechten Seite von Zuweisungen stehen, wogegen Bedingungen nur in bedingten Sprunganweisungen vorkommen.

3.11.1. Arithmetische Ausdrücke.

Ein arithmetischer Ausdruck verknüpft zwei Operanden mit den Operatoren '+', '-' und '*'.

arithausdruck::= operand [arithoperator operand]
arithoperator::= + | - | *

Anmerkung: Die unter 2.3.1 angegebene Verträglichkeitstabelle schränkt die Verwendung von Operanden ein. Z.B. ist rechts vom Multiplikationszeichen kein POINTER und damit auch keine Adreßkonstante erlaubt.

3.11.2. Bedingungen

Bedingungen verknüpfen einen arithmetischen Ausdruck (bzw. auch eine Zuweisung) über einen Vergleichsoperator mit einem Operanden und liefern den Wahrheitswert TRUE oder FALSE ab.

bedingung::=
{zuweisungsausdruck | arithausdruck} vergleich | selektion

vergleich::=
vergleichsoperator operand | .NIL. | .NULL.

vergleichsoperator::=
.EQ.|.NE.|.GT.|.LT.|.GE.|.LE.

selektion::= operand (zahl)

Anmerkung: Die Zeichenfolgen .NIL. bzw. .NULL. stellen Abkürzungen der Zeichenfolgen .EQ. $NIL bzw. .EQ. 'KØ' dar.
Bei der Selektion muß der Operand vom Typ BITKETTE sein.

3.12. Zuweisung

Es sind Einfach- und Mehrfachzuweisungen vorgesehen

zuweisungsausdruck::=
{{register | speicherzelle} := }... arithausdruck

zuweisung::=
zuweisungsausdruck; eol

3.13. Sprungbefehle

sprung::=
unbedingter sprung | bedingter sprung | unterprogrammsprung | rücksprung

3.13.1. Unbedingter Sprung

unbedingter sprung::=
␣.TO. label; eol
label::= adresse | substitution

3.13.2. Bedingter Sprung

bedingtersprung::=
␣.TO. label {.IF. | .IFNOT.} bedingung; eol

3.13.3. Unterprogrammsprung

unterprogrammsprung::=
␣.CALL. label; eol

Anmerkung: Die Adresse, die sich bei Auswertung von 'label' ergibt, muß eine Unterprogrammarke sein (siehe 3.15)

3.13.4. Rücksprung

rücksprung::=
␣.RETURN. adresse; eol

Anmerkung: Die 'adresse' muß die zugehörige Unterprogrammarke sein.

3.14. Organisatorische Befehle

orgbefehl::=
save | restore | disable | enable | einausgabe | fehler | halt

3.14.1. Retten der Register eines Rechenprozesses

Es wird vorausgesetzt, daß bei der Ausführung des Befehles der gerade arbeitende Rechenprozeß - und damit der zugehörige Rettbereich - bekannt ist.

save::= ␣SAVEREGISTERS; eol |
␣SAVEALLREGISTERS; eol

Die erste Alternative wird verwendet, wenn sich ein Rechenprozeß selbst durch einen Betriebssystemaufruf unterbricht.

Die zweite Alternative wird für Unterbrechungen durch Interrupts benötigt.

3.14.2. Register eines Rechenprozesses restaurieren und Rechenprozeß anspringen.

Es wird vorausgesetzt, daß vor Ausführung des Befehles in irgendeiner Form ein Rechenprozeß ausgewählt wurde, daß also Registerrettbereich - und damit auch der zugehörige Befehlszähler stand - bekannt sind. Eine Interruptfreigabe ist impliziert.

restore::= ␣RESUMEPROCESS; eol

3.14.3. Interrupt sperren

disable::= ␣DISABLE; eol

3.14.4. Interrupt freigeben

enable::= ␣ENABLE; eol

3.14.5. Geräteversorgung

Es wird vorausgesetzt, daß zuvor in irgendeiner Form die Versorgungsparameter eines Gerätes identifiziert wurden (z.B. über ein Register, ... auf einen Versorgungsblock zeigt). Der Befehl selbst stößt dann die betreffende Ein- oder Ausgabe an.

einausgabe::= ␣DOIO; eol

3.14.6. Fehlerstop

Dieser Befehl wird bei nicht behebbaren Fehlersituationen verwendet. Seine Folgen sind undefiniert.

fehler::=
␣.ERROR.zahl 'name'; eol

Anmerkung: Die Zahl und der Name dienen nur zur Identifizierung des Fehlers.

3.14.7. Warten auf Interrupt

Dieser Befehl stellt ein 'aktives Warten' dar; d.h. der Befehlsablauf ist gestoppt - ankommende Interrupts bewirken jedoch ein Anspringen der vorgesehenen Unterbrechungsroutinen.

halt::= ␣HALT; eol

3.15. Prozeduren

Prozeduren können über Unterprogrammsprünge aufgerufen, und über Rücksprünge verlassen werden. Parameterübergaben erfolgen nur über Register, wobei keine Registerrettmechanismen vorgesehen sind.

```
prozedur::=
  ␣PROC : adresse; eol
{markenvereinbarung | zuweisung | sprung | orgbefehl}···
  ␣END : adresse; eol
```

Anmerkung: Die erste Zeile definiert die Ansprungadresse der Prozedur. Die 'adresse' in der letzten Zeile muß mit der Ansprungadresse identisch sein.

3.16. Programm

```
programm::=
   ␣START; eol
  längenvereinbarung ···
 [strukturvereinbarung ···]
 {speicherreservierung | markenvereinbarung | zuweisung |
  sprung | orgbefehl | prozedur}···
 ␣FINISH; eol
```

4. Alphabetisches Verzeichnis der benutzten Metasprachvariablen

In der zweiten Spalte (unter 'Definition') ist die Nummer des Abschnitts zu finden, in dem die Variable definiert wird. In der dritten Spalten (unter 'Verwendung') sind die Nummern der Abschnitte angeführt, in denen die Variable verwendet wird.

Variable	Definition	Verwendung
adresse	3.5.2.	3.5.2, 3.6, 3.7, 3.9, 3.13.1, 3.13.4, 3.15
adreßkonstante	3.5.2	3.5
arithausdruck	3.11.2	3.11.2, 3.12
arithoperator	3.11.1	3.11.1
bedingtersprung	3.13.2	3.13
bedingung	3.11.2	3.13.2
binärzahl	-	3.5.3
bitkettenkonstante	3.5.3	3.5
disable	3.14.3	3.14
einausgabe	3.14.5	3.14
elementartyp	3.1	3.2, 3.3
enable	3.14.4	3.14
eol	3.3	3.3-3.16 (mit Ausnahmen)
fehler	3.14.6	3.14
halt	3.14.7	3.14
indizierung	3.9	3.9
integerkonstante	3.5.1	3.5
kommentar	-	3.3
konstante	3.5	3.6, 3.10
label	3.13.1	3.13.1, 3.13.2, 3.13.3
längenvereinbarung	3.3	3.16
markenvereinbarung	3.7	3.15, 3.16
name	-	3.2, 3.4, 3.5.2, 3.14.6
offset	3.4	3.4, 3.5.2, 3.9
operand	3.10	3.11.1, 3.11.2
orgbefehl	3.14	3.15, 3.16
programm	3.16	
prozedur	3.15	3.16
register	3.8	3.9, 3.10, 3.12
restore	3.14.2	3.14

Variable	Definition	Verwendung
rücksprung	3.14	3.13
save	3.14.1	3.14
selektion	3.11.2	3.11.2
speicherreservierung	3.6	3.16
speicherzelle	3.9	3.10, 3.12
sprung	3.13	3.15, 3.16
strukturtyp	3.2	3.2, 3.4
strukturvereinbarung	3.4	3.16
substitution	3.9	3.9, 3.13.1
typ	3.2	3.4, 3.6
unbedingtersprung	3.13.1	3.13
universalregister	3.8	3.8, 3.9
unterprogrammsprung	3.13.3	3.13
vergleich	3.11.2	3.11.2
vergleichsoperator	3.11.2	3.11.2
zahl	-	3.3, 3.5.1, 3.6, 3.14.6
zuweisung	3.12	3.13, 3.16
zuweisungsausdruck	3.12	3.11.2, 3.12

Syntax von SPASS

```
programm::=  START; eol
              längenvereinbarung ···
             [strukturvereinbarung ··· ]
             {speicherreservierung | markenvereinbarung | zuweisung |
              sprung | orgbefehl | prozedur }···
             FINISH; eol

prozedur::=  PROC : adresse; eol
             {markenvereinbarung | zuweisung | sprung | orgbefehl }···
             END : adresse; eol

orgbefehl::={SAVE REGISTERS | SAVE ALL REGISTERS |
             RESUME PROCESS | DISABLE | ENABLE | DOIO | HALT |
              .ERROR. zahl 'name' } ; eol

sprung::=    {.TO. label |
              .TO. label {.IF. | .IFNOT.} bedingung |
              . CALL. label |
              . RETURN. adresse } ; eol

label::=      adresse | substitution

zuweisung::=  zuweisungsausdruck; eol

zuweisungsausdruck::= {{register | speicherzelle} :=}···  arithausdruck

bedingung::= { zuweisungsausdruck | arithausdruck } vergleich | selektion

vergleich::=   vergleichsoperator  operand | .NIL. | .NULL.

vergleichsoperator::=  .EQ. | .NE. | .GT. | .LT. | .GE. | .LE.

selektion::=  operand (zahl)

arithausdruck::=  operand [arithoperator  operand ]
```

```
arithoperator::=  + | - | *

operand::=        konstante | register | speicherzelle

speicherzelle::=  adresse | indizierung | substitution

indizierung::=    offset, universalregister

substitution::=   $Ø, adresse | $Ø, register

register::=       universalregister | R4 | R5 | R6 | R7

universalregister::=   R1 | R2 | R3

markenvereinbarung::=  LOC  : adresse ; eol

speicherreservierung::=  SPACE : adresse = [zahl *] typ [+] ; eol
       [[{[zahl*] SPEL = konstante + ; eol }··· ]

       [zahl *] SPEL = konstante; eol ]

adresse::= name

konstante::= 'Kzahl' | 'KMzahl' | $ adresse | $ offset |
             $NIL | 'B binärzahl'

strukturvereinbarung::= STRUCT : strukturtyp; eol
                        [{ELEM : offset = typ; eol }··· ]
                          STREND : strukturtyp; eol

offset::= name

längenvereinbarung::= LENGTH : elementartyp = zahl; eol

eol::= kommentar EOL [{/kommentar EOL ··· }]

strukturtyp::= name

typ::= elementartyp | strukturtyp

elementartyp::=  INT | POINTER | BITS
```

Nicht produzierte non-terminals:

name
zahl
binärzahl
kommentar

EOL bedeutet 'Zeilenendezeichen' bzw. Ende einer Lochkarte.

```
START;
/                               SPASS-PEARL-BETRIEBSSYSTEM;
/                               ==========================;
/;
/                               RUFBUS-SUBSET;
/                               ROLAND ROESSLER;
/;
/                               REGISTERZUORDNUNGEN;
/;
/                               R1=IX;
/                               R2=IY;
/                               R3=BC;
/                               R4,R5,R6,R7..SPEICHERZELLEN;
/;
   LENGTH: POINTER=2;                    TYPLAENGENVEREINBARUNGEN
   LENGTH: INT=1;
/;
   STRUCT:CHAIN;                         WARTESCHLANGENELEMENT
      ELEM:SUCC=POINTER;                 ZEIGER AUF NACHFOLGER
      ELEM:PRED=POINTER;                 ZEIGER AUF VORGAENGER
   STREND:CHAIN;
/;
   STRUCT:PARBLOC;                       AUFRUF-PARAMETERBLOCK
      ELEM:SKETT=POINTER;                SCHEDULE-KETTE
      ELEM:SPCB=POINTER;                 ZEIGER AUF TCB,SEMA
      ELEM:PSVC=POINTER;                 ZEIGER AUF ROUTINE
      ELEM:STYP=INT;                     SCHEDULE-TYP
   STREND:PARBLOC;
/;
   STRUCT:SCHED;                         SCHEDULE
      ELEM:SCPAR=PARBLOC;                PARAMETERBLOCK
      ELEM:SCPRIO=INT;                   'SCHEDULE'-PRIORITAET
      ELEM:SBEC=INT;                     INTERRUPT-NUMMER
      ELEM:TYP=INT;                      AKTUELLER TYP
      ELEM:SCHART=POINTER;               ZEIGER AUF ZEIGER AUF SCHEDULE
   STREND:SCHED;
/;
   STRUCT:SEMA;                          SEMAPHORE
      ELEM:SWS=CHAIN;                    SWS
      ELEM:PRIO=INT;                     WERT
   STREND:SEMA;
/;
   STRUCT:PCB;                           TASKKONTROLLBLOCK
      ELEM:KOPF=SEMA;                    KOPF
      ELEM:SCHPNT=POINTER;               ZEIGER AUF ACTIVATE-SCHEDULE
      ELEM:SCHRES=POINTER;               ZEIGER AUF CONTINUE-SCHEDULE
      ELEM:STATUS=INT;                   ZUSTANDSKENNUNG
      ELEM:PC=POINTER;                   AKT. BEFEHLSZAEHLER
      ELEM:SP=POINTER;                   AKT.STACKPOINTER
      ELEM:STPC=POINTER;                 STARTADRESSE
      ELEM:STSP=POINTER;                 STARTSP
   STREND:PCB;
```

```
/;
/;
/                              INITIALISIERUNGSROUTINE;
/                              =======================;
/;
/;
 LOC:INIT;
        R1:=$RAMANF;                        ANFADR RAM
    LOC:IN01;
        .TO. IN02 .IF. R1 .GE. $RAMEND;
        $0,R1:=$NIL;                        RAM LOESCHEN
        R1:=R1+$POINTER;
        .TO. IN01;
    LOC:IN02;
        R1:=$INLIST;                        TCB'S INITIALISIEREN
    LOC:VOR;
        .TO. FERTIG .IF. R4:=$0,R1 .NIL.;   STARTADRESSE
        R1:=R1+$POINTER;
        R2:=$0,R1;                          TCB-ADRESSE
        R1:=R1+$POINTER;
        R3:=$POINTER*'K6';                  ANZAHL REGISTER IM STACK
        STSP,R2:=$0,R1-R3;                  SP-12 IN TCB
        STPC,R2:=R4;                        STARTADR. IN TCB
        R3:=R2+$PCB;                        ADR. DES ACT-SCHEDULE
        SPCB,R3:=R2;                        TCB-ADR IN SCHEDULE
        R3:=R3+$SCHED;                      ADR. DES CONT.-SCHEDULE
        SPCB,R3:=R2;
        R1:=R1+$POINTER;
        .TO. VOR;
    LOC:FERTIG;
        R1:=$ITRLIST;                       ITR-LISTE INITIALISIEREN
        R5:=ILANG;                          LAENGE DER LISTE
        .CALL. INI1;
        R1:=RUNNING:=$RUNNING;              PWS INITIALISIEREN
        PRED,R1:=$RUNNING;
        R1:=$ORGSCH;
        .TO. SWC0;                          INITIALTASK STARTEN
/;
/;
    PROC:INI1;                              INITIALISIEREN EINER LISTE
    LOC:INI2;                               R1:ADRESSE, R5:LAENGE
        $0,R1:=R1;                          ELEMENT:=ZEIGER AUF SICH SELBST
        R1:=$POINTER+R1;
        .TO. INI2 .IFNOT. R5:=R5-'K1' .NULL.;
        .RETURN. INI1;
    END:INI1;
```

```
/;
/;
/                              SUPERVISOR-CALL;
/                              ===============;
/;
/                              WIRD VON ALLEN BS-AUFRUFEN BENUTZT;
/                              VERSORGUNG: R1..ZEIGER AUF PARBLOCK BZW.SCHED;
/;
 LOC:SVC;
          SAVE REGISTERS;
     LOC:SVCØ;
          R2:=SPCB,R1;                        ZEIGER AUF TCB ODER SEMA
          .TO. SVC1 .IF. R4:=STYP,R1 .NULL.; KEIN SCHEDULE
          TYP,R1:=R4;                         TYP INITIALISIEREN
     LOC:SVC1;
          R3:=PSVC,R1;                        ZEIGER AUF ROUTINE
          .CALL. $Ø,R3;
     LOC:ASSIGN;
          .TO. NOTASK .IF. $RUNNING .EQ. RUNNING;
          RESUME PROCESS;                     TASK AUFNEHMEN
     LOC:NOTASK;
          ENABLE;
          HALT;                               WARTEN AUF ITR
/;
/;
/                              INTERRUPT;
/                              =========;
/;
/                              VERSORGUNG: R1..INTERRUPTNUMMER(Ø..19);
/;
 LOC:ITR;
          R1:=$POINTER*R1;
          THZ:=R1:=$ITRLIST+R1;               ADR.DER ITR-ZELLE
          THZ1:=$Ø,R1;                        INHALT DER ITR-ZELLE
     LOC:TIM3;                                EINGEKETTETE SCHEDULES SUCHEN
          .TO. ASSIGN .IF. R1:=THZ1 .EQ. THZ;
          THZ1:=" R1;
          R2:=SPCB,R1;
          .CALL. SCH;                         SCHEDULE-VERARBEITUNG
          .TO. TIM3 .IF. R3 .NIL.;            KEINE OPERATION AUSZUFUEHREN
          .CALL. $Ø,R3;
          .TO. TIM3;                          WEITERSUCHEN
```

```
/;
/;
/                              ACTIVATE;
/                              ========;
/;
/                              VERSORGUNG:    R1...ZEIGER AUF SCHEDULE;
/                                             R2...ZEIGER AUF TCB;
/                                             R4...SCHEDULE-TYP;
/;
 PROC:START;
          .TO. STA1 .IF. R4 .NULL.;       KEIN SCHEDULE
          SCHART,R1:=R2+$SCHPNT;
          .CALL. SCH;                     SCHEDULE-VERARBEITUNG
          .RETURN. START;
     LOC:STA1;
          .TO. LOOK .IFNOT. STATUS,R2 .NULL.;
          PRIO,R2:=SCPRIO,R1;
          STATUS,R2:='B1000';
          PC,R2:=STPC,R2;                 PC INITIALISIEREN
          SP,R2:=STSP,R2;                 SP INITIALISIEREN
          R1:=$RUNNING;           IN RUNNING-KETTE EINKETTEN
          .CALL. CHAIN;
     LOC:LOOK;
          .RETURN. START;
 END:START;
/;
/;
/                              REQUEST;
/                              =======;
/;
/                              VERSORGUNG: R2...ZEIGER AUF SEMA;
/;
 PROC:REQU;
          R1:=R2;                         SEMA-ADRESSE
          .TO. REQ0 .IFNOT. SUCC,R1 .NIL.;
          SUCC,R1:=PRED,R1:=R1;
     LOC:REQ0;
          R2:=RUNNING;
          .TO. REQU1 .IF. PRIO,R1 .NULL.;
          PRIO,R1:=PRIO,R1-'K1';
          .RETURN. REQU;
     LOC:REQU1;
          STATUS,R2:='B0001';             TASK UNTERBRECHEN
          .CALL. UNCH;                    AUSKETTEN AUS PWS
          .CALL. CHAIN;                   EINKETTEN IN SWS
          .RETURN. REQU;
 END:REQU;
```

```
/;
/;
/                               RELEASE;
/                               ========;
/;
/                               VERSORGUNG: R2...ZEIGER AUF SEMA;
/;
 PROC:RELE;
          R1:=R2;                              SEMA-ADRESSE
          .TO. RELØ .IFNOT. SUCC,R1 .NIL.;
          SUCC,R1:=PRED,R1:=R1;
     LOC:RELØ;
          .TO. REL2 .IF. R2:=$Ø,R1 .EQ. R1;  KETTE LEER
          R1:=$RUNNING;
          .CALL. UNCH;                         AUS SEMA-KETTE
          .CALL. CHAIN;                        IN RUNNING-KETTE
          STATUS,R2:='B1ØØØ';
          .RETURN. RELE;
     LOC:REL2;
          PRIO,R1:=PRIO,R1+'K1';
          .RETURN. RELE;
 END:RELE;
/;
/;
/                               TERMINATE;
/                               =========;
/;
/                               VERSORGUNG: R2...ZEIGER AUF TCB ODER Ø;
/;
 PROC:STOP;
          .TO. STOP4 .IFNOT. R2 .NIL.;  NICHT SELBST
          R2:=RUNNING;
     LOC:STOP1;
          .CALL. UNCH;                         AUS PWS
     LOC:STOP2;
          STATUS,R2:='KØ';
     LOC:STOP3;
          .RETURN. STOP;
          LOC:STOP4;
          .TO. STOP3 .IF. STATUS,R2 .NULL.;        WENN BEREITS BEENDET
          .TO. STOP2 .IF. STATUS,R2 .EQ. 'B0100'; WENN SUSPENDIERT
          .TO. STOP1;                              WENN AKTIV ODER BLOCKIERT
 END:STOP;
```

```
/;
/;
/                                PREVENT;
/                                ========;
/;
/                                VERSORGUNG: R2...ZEIGER AUF TCB ODER Ø;
/;
 PROC:PREV;
          .TO. PREV1 .IFNOT. R2 .NIL.;
          R2:=RUNNING;
     LOC:PREV1;
          R1:=SCHPNT,R2;                    ACT-SCHEDULE ELIMINIEREN
          .CALL. DEQU;
          SCHPNT,R2:=$NIL;
          R1:=SCHRES,R2;                    RESUME-SCHEDULE ELIMINIEREN
          .CALL. DEQU;
          SCHRES,R2:=$NIL;
          .RETURN. PREV;
 END:PREV;
/;
/;
/                                SUSPEND;
/                                ========;
/;
/                                VERSORGUNG: KEINE;
/;
 PROC:SUSP;
          R2:=RUNNING;
          STATUS,R2:='BØ1ØØ';
          .CALL. UNCH;
          .RETURN. SUSP;
 END:SUSP;
```

```
/;
/;
/                              CONTINUE;
/                              ========;
/;
/                              VERSORGUNG:     R1...ZEIGER AUF SCHEDULE;
/                                              R2...ZEIGER AUF TCB ODER Ø;
/                                              R4...SCHEDULE-TYP;
/;
 PROC:CONT;
          .TO. CONTØ .IFNOT. R2 .NIL.;
          R2:=RUNNING;
     LOC:CONTØ;
          .TO. CONT1 .IF. R4 .NULL.;     KEIN SCHEDULE
          SCHART,R1:=R2+$SCHRES;
          .CALL. SCH;
          .RETURN. CONT;
     LOC:CONT1;
          .TO. CONT2 .IF. STATUS,R2 .NE. 'BØ1ØØ';
          STATUS,R2:='B1ØØØ';
          R1:=$RUNNING;
          .CALL. CHAIN;
     LOC:CONT2;
          .RETURN. CONT;
 END;CONT;
/;
/;
/                              RESUME;
/                              ======;
/;
/                              VERSORGUNG:     R1...ZEIGER AUF SCHEDULE;
/                              R2...Ø;
/                              R4...SCHEDULE-TYP;
/;
 PROC;RESU;
          .TO. RESU1 .IF. R4 .NULL.;
          SPCB,R1:=R2:=RUNNING;
          .CALL. CONT;
          .CALL. SUSP;
          .RETURN. RESU;
     LOC:RESU1;
          .CALL. CONT;
          .RETURN. RESU;
 END;RESU;
```

```
/;
/;
/                              EINKETTEN;
/                              ==========;
/;
/                              VERSORGUNG:     R1...ZEIGER AUF KETTENANFANG;
/                                              R2...ZEIGER AUF ELEMENT;
/;
     PROC:CHAIN;
          R5:=R1;
          R3:=SUCC,R1;                        NACHFOLGER
          R6:=PRIO,R2;                        PRIORITAET
     LOC:LOOP;
          .TO. IN .IF. R3 .EQ. R5;            KETTENENDE
          .TO. IN .IF. R6 .LT. PRIO,R3;  HOEHERE PRIO
          R1:=SUCC,R1;
          R3:=SUCC,R1;
          .TO. LOOP;
     LOC:IN;
          SUCC,R1:=PRED,R3:=R2;
          PRED,R2:=R1;
          SUCC,R2:=R3;
          .RETURN. CHAIN;
     END:CHAIN;
/;
/;
/                              AUSKETTEN;
/                              ==========;
/;
/                              VERSORGUNG: R2..ZEIGER AUF ELEMENT;
/;
     PROC:UNCH;
          R4:=R3:=PRED,R2;
          R3:=SUCC,R3:=SUCC,R2;
          PRED,R3:=R4;
          .RETURN. UNCH;
     END:UNCH;
```

```
/;
/;
/                              AUSKETTEN AUS SCHEDULE-KETTE;
/                              ============================;
/;
/                              VERSORGUNG: R1...ZEIGER AUF SCHED. ODER Ø;
/;
     PROC:DEQU;
          .TO. DEQ1 .IFNOT. R1 .NIL.;
          .RETURN. DEQU;                    KEIN ELEMENT AUSZUKETTEN
     LOC:DEQ1;
          R3:=R1;
          .TO. DEQ2 .IFNOT. R4:=SKETT,R1 .NIL.;
          .RETURN. DEQU;                    ELEMENT IST IN KEINER KETTE
     LOC:DEQ2;
          R3:=SKETT,R3;
          .TO. DEQ2 .IF. SKETT,R3 .NE. R1;
          SKETT,R3:=R4;
          SKETT,R1:=$NIL;
          .RETURN. DEQU;
     END:DEQU;
/;
/;
/                              SCHEDULE-VERARBEITUNG;
/                              =====================;
/;
/                              VERSORGUNG: R1...ZEIGER AUF SCHED;
/;
     PROC:SCH;
          .TO. SCHOP .IF. TYP,R1 .EQ. 'K4';  WENN ABGELAUFEN
          R5:=R1;
          R3:=R6:=SCHART,R1;                ZEIGER AUF 'SCHPNT' ODER 'SCHRES'
          R1:=$Ø,R3;                        INHALT VON              -''-
          .CALL. DEQU;                      ALTEN SCHEDULE AUSKETTEN
          R3:=R6;
          $Ø,R3:=R1:=R5;                    IN 'SCHPNT' ODER 'SCHRES' SCHED.ADR.
          TYP,R1:=TYP,R1-'K8';
          R2:=$POINTER*SBEG,R1;
          R2:=%ITRLIST+R2;                  ZEIGER AUF ITR-ZELLE
          SKETT,R1:=$Ø,R2;
          $Ø,R2:=R1;                        SCHEDULE EINKETTEN
     LOC:SCH3;
          R3:=$NIL;                         KENNUNG: KEINE FOLGEOPERATION
          .RETURN. SCH;
     LOC:SCHOP;
          R3:=PSVC,R1;                      ADR. DER FOLGEOPERATION
          R4:='KØ';
          .RETURN. SCH;
     END:SCH;
```

```
/;
/;
/                                   DATEN;
/                                   =====;
/;
     SPACE:ORGSCH=SCHED+;                        INITIALSCHEDULE
           SPEL=$NIL+;
           SPEL=$TCB1+;
           SPEL=$START+;
           SPEL='KØ'+;
           SPEL='KØ';
/;
     SPACE:ILANG=INT+;                           LAENGE DER ITR-LISTE
           SPEL='K2Ø';
/                                   RAM-DATEN;
/                                   =========;
/;
     SPACE:ITRLIST=2Ø*POINTER;                   INTERRUPTLISTE
/;
     SPACE:RUNNING=PCB;                          PWS-KOPF
/;
     SPACE:THZ=POINTER;
     SPACE:THZ1=POINTER;
/;
/;
 FINISH;
```

```
*                    Z80-PEARL-BETRIEBSSYSTEM
*                    ========================
*                    (ANLAGENABHAENGIGER TEIL)
*
*                    ANWEISUNGEN MIT $ IN SPALTE 80 SIND NICHT ENDGUELTIG
*
*                    INITIALISIERUNG (UEBER 'POWER ON'/'RESET')
*
ANFANG   LD    SP,ORGST      *STACKPOINTER:= BS-STACK
         IM    1             *(BZW. I-REG LADEN, IM 2)
         EX    AF,AF'        *
         EXX                 *ZWEITER REGISTERSATZ FUER SCHNELLE REAKTION
         XOR   A             *A:= MERKER 'BS TAETIG'
         LD    HL,ISPEI      *HL:= INTERRUPTKELLER
         LD    (HL),0        *1.PLATZ LOESCHEN
         EXX                 *
         EI                  *
         EX    AF,AF'        *
         LD    IY,TCB1       *R2:= ADR. DES 1. TCB
         JP    INIT          *ZUM ANLAGENUNABHAENGIGEN TEIL
*
*
*                    SCHNITTSTELLE FUER BS-AUFRUFE OHNE 'SCHEDULE'
*
*                    AUFRUFSEQUENZ:
*                              CALL BS
*                              DW   TCB/0/SEMA
*                              DW   OPERATION
*                              DB   0
*                              DB   PRIORITAET/0
*
BS       EX    (SP),HL       *ADR. DES 1. PARAMETERS
         PUSH  AF            *
         PUSH  BC            *
         PUSH  DE            *
         DEC   HL            *
         PUSH  IX            *
         DEC   HL            *
         PUSH  HL            *
         POP   IX            *ADR. 'PARBLOC'
         LD    DE,8D         *
         ADD   HL,DE         *RUECKSPRUNGADR.
         DI                  *
         EX    AF,AF'        *
         XOR   A             *A:= MERKER 'BS TAETIG'
         EI                  *
         EX    AF,AF'        *
SAVE1    PUSH  IY            *
         LD    IY,(RUNNING)  *ADR. TCB
         LD    (IY+10D),L    *PC RETTEN
         LD    (IY+11D),H    *
         LD    HL,0          *
         ADD   HL,SP         *
         LD    (IY+12D),L    *SP RETTEN
         LD    (IY+13D),H    *
         LD    SP,ORGST      *
         JP    SVC0          *ZUM ANL.UNABH. TEIL
*
*
*                    SCHNITTSTELLE FUER SCHEDULE-AUFRUFE
*
*                    AUFRUFSEQUENZ:
*                              CALL SCHED
*                              DW   TCB+18D   BEI ACTIVATE
```

```
*                       (BZw.  DW    TCB+30D    BEI CONTINUE
*                        BZw.  DW    30D        BEI RESUME)
*                              DW    OPERATION
*                              DB    12D
*                              DB    PRIORITAET/0
*                              DB    INTERRUPTNUMMER
*
SCHED   EX    (SP),HL          *ADR. DES 1.PARAMETERS
        PUSH  AF               *
        PUSH  BC               *
        PUSH  DE               *
        PUSH  IX               *
        LD    E,(HL)           *
        XOR   A                *
        INC   HL               *
        LD    D,(HL)           *DE: SCHEDULE-ADRESSE
        OR    D                *
        JR    NZ,SC1           *WENN NICHT AUFRUFENDE TASK
        EX    DE,HL            *
        LD    BC,(RUNNING)     *
        ADD   HL,BC            *HL: SCHEDULEADRESSE
        EX    DE,HL            *
SC1     INC   HL               *
        PUSH  DE               *
        POP   IX               *R1:=SCHEDULE-ADRESSE
        INC   DE               *
        INC   DE               *
        INC   DE               *
        INC   DE               *
        LD    BC,5             *
        DI                     *
        EX    AF,AF'           *
        XOR   A                *KENNUNG 'BS TAETIG'
        EI                     *
        EX    AF,AF'           *
        LDIR                   *PARAMETER IN SCHEDULE UMSPEICHERN
        JR    SAVE1            *
*
*               ROUTINEN ZUR SIMULATION VON BC ALS INDEXREGISTER
*
*
*               LADEN UEBER BC MIT OFFSET EINFACH
*
R3L1    LD    L,A              *OFFSET
        LD    H,0              *
        ADD   HL,BC            *HL:=(BC)+OFFSET
        LD    A,(HL)           *
        RET                    *
*
*               LADEN UEBER BC MIT OFFSET DOPPELT
*
R3L2    PUSH  DE               *
        LD    L,A              *
        LD    H,0              *
        ADD   HL,BC            *HL:=(BC)+OFFSET
        LD    E,(HL)           *
        INC   HL               *
        LD    D,(HL)           *
        EX    DE,HL            *
        POP   DE               *
        RET                    *
*
*               SPEICHERN UEBER BC MIT OFFSET DOPPELT
*
R3S2    LD    D,0              *
        EX    DE,HL            *
```

```
        ADD   HL,BC           *HL:=(BC)+OFFSET
        LD    (HL),E          *
        INC   HL              *
        LD    (HL),D          *
        EX    DE,HL           *
        RET                   *
*
*                SIMULATION DER IM2-REAKTION                          $
*                                                                     $
I2SIM   PUSH  AF              *                                       $
        PUSH  HL              *                                       $
        PUSH  BC              *                                       $
        IN    2               *                                       $
        AND   3               *                                       $
        ADD   A               *ITR-NUMMER*2                           $
        LD    HL,VEKTOR       *                                       $
        LD    B,0             *                                       $
        LD    C,A             *                                       $
        ADD   HL,BC           *                                       $
        LD    C,(HL)          *                                       $
        INC   HL              *                                       $
        LD    B,(HL)          *                                       $
        LD    (SPR+1),BC      *                                       $
        POP   BC              *                                       $
        POP   HL              *                                       $
        POP   AF              *                                       $
SPR     JP    0               *                                       $
*                                                                     $
*                INTERRUPTVEKTOR                                      $
*                                                                     $
VEKTOR  DW    ITR0            *                                       $
        DW    ITR1            *                                       $
        DW    ITR2            *                                       $
        DW    ITR3            *                                       $
*       .                                                             $
*       .                                                             $
*       .                                                             $
*
*                INTERRUPT-SERVICE-ROUTINEN
*                (FUER INTERRUPTS, DIE AN DAS BS WEITERGEREICHT WERDEN)
*
ITR0    EX    AF,AF'          *
        SRL   A               *MERKER TESTEN
        JR    C,7+$           *WENN ANWENDERTASK UNTERBROCHEN
        EI                    *WENN BS UNTERBROCHEN
        EX    AF,AF'          *
        PUSH  AF              *
        LD    A,1             *ITR-NUMMER FUER KELLER (BEL.ZW. 1..19)   $
        JR    ISAVE           *
        EI                    *
        EX    AF,AF'          *
        PUSH  AF              *
        LD    A,1             *ITR-NUMMER FUER BS                       $
        JR    ANW             *
ITR1    EX    AF,AF'          *
        SRL   A               *MERKER TESTEN
        JR    C,7+$           *WENN ANWENDERTASK UNTERBROCHEN
        EI                    *WENN BS UNTERBROCHEN
        EX    AF,AF'          *
        PUSH  AF              *
        LD    A,2             *ITR-NUMMER FUER KELLER (BEL.ZW. 1..19)   $
        JR    ISAVE           *
        EI                    *
        EX    AF,AF'          *
        PUSH  AF              *
        LD    A,2             *ITR-NUMMER FUER BS                       $
```

```
        JR    ANW               *
ITR2    EX    AF,AF'            *
        SRL   A                 *MERKER TESTEN
        JR    C,7+$             *WENN ANWENDERTASK UNTERBROCHEN
        EI                      *WENN BS UNTERBROCHEN
        EX    AF,AF'            *
        PUSH  AF                *
        LD    A,3               *ITR-NUMMER FUER KELLER (BEL.ZW. 1..19)
        JR    ISAVE             *
        EI                      *
        EX    AF,AF'            *
        PUSH  AF                *
        LD    A,3               *ITR-NUMMER FUER BS
        JR    ANW               *
ITR3    EX    AF,AF'            *
        SRL   A                 *MERKER TESTEN
        JR    C,7+$             *WENN ANWENDERTASK UNTERBROCHEN
        EI                      *WENN BS UNTERBROCHEN
        EX    AF,AF'            *
        PUSH  AF                *
        LD    A,4               *ITR-NUMMER FUER KELLER (BEL.ZW. 1..19)
        JR    ISAVE             *
        EI                      *
        EX    AF,AF'            *
        PUSH  AF                *
        LD    A,4               *ITR-NUMMER FUER BS
        JR    ANW               *
*                               *
*               INTERRUPTS ZWISCHENSPEICHERN
*                 (WENN BS UNTERBROCHEN)
*
ISAVE   DI                      *
        EXX                     *
        INC   HL                *
        LD    (HL),A            *ITR KELLERN
        EI                      *
        EXX                     *
        POP   AF                *
        RETI                    *
*
*               TESTEN AUF GESPEICHERTE INTERRUPTS
*
TEST    XOR   A                 *A:=0
        DI                      *
        EXX                     *
        OR    (HL)              *ITR-NUMMER IM AKKU
        JR    Z,KEINIT          *ACHTUNG: 0 IST 'KEIN INTERRUPT'
        DEC   HL                *NAECHSTER KELLERPLATZ
        EI                      *
        EXX                     *
        LD    (INR),A           *INTERRUPTNUMMER
        JR    ANW0              *
KEINIT  EI                      *
        EXX                     *
        RET                     *
*
*               REAKTION AUF 'TASK UNTERBROCHEN'
*
ANW     LD    (INR),A           *INTERRUPTNUMMER
        POP   AF                *
        EX    (SP),HL           *PC UND HL TAUSCHEN
        PUSH  AF                *
        LD    (PC),HL           *PC RETTEN
        LD    HL,RSAVE          *
        PUSH  HL                *
        RETI                    *AUCH NIEDRIGPRIORE DURCHLASSEN
```

```
RSAVE   PUSH BC             *ALLE REGISTER RETTEN
        PUSH DE             *
        PUSH IX             *
        PUSH IY             *
*
*                   ANORDNUNG IM STACK:
*                              IY
*                              IX
*                              DE
*                              BC
*                              AF
*                              HL
*
        LD   IY,(RUNNING)   *STACKPOINTER RETTEN
        LD   HL,0           *
        ADD  HL,SP          *
        LD   (IY+12D),L     *
        LD   (IY+13D),H     *
        LD   HL,(PC)        *PC RETTEN
        LD   (IY+10D),L     *
        LD   (IY+11D),H     *
ANWO    LD   IX,(INR)       *R1:=INTERRUPTNUMMER
        LD   SP,ORGST       *
        JP   ITR            *ZUM ANLAGENUNABHAENGIGEN TEIL
*
*                  'RESUME PROCESS'
*
RESUMP  LD   IY,(RUNNING)   *ADR. TCB
        LD   L,(IY+12D)     *SP-STAND
        LD   H,(IY+13D)     *
        LD   SP,HL          *
        LD   L,(IY+10D)     *PC-STAND
        LD   H,(IY+11D)     *
        CALL TEST           *TESTEN OB NOCH INTERRUPTS GESPEICHERT
        POP  IY             *
        POP  IX             *
        POP  DE             *
        POP  BC             *
        POP  AF             *
        EX   (SP),HL        *
        DI                  *
        EX   AF,AF'         *
        LD   A,1            *A:= MERKER 'ANWENDERTASK'
        EX   AF,AF'         *
        EI                  *
        RET                 *
*
*                   'ENABLE'
ENABL   LD   SP,ORGST       *SP AUF BS-STACKBEREICH
        CALL TEST           *
        DI                  *
        EX   AF,AF'         *
        LD   A,1            *A:= MERKER 'ANWENDERTASK'
        EI                  *
        EX   AF,AF'         *
        HALT                *
*
*                   RAM-BEREICH
*
RAMANF  EQU  200000         *
RAMEND  EQU  205000         *
TCB1    EQU  RAMANF+1000    *ADRESSE DES TCB DER INITIALTASK
*
PC      DW   0              *
INR     DW   0              *
ISPEI   DS   100            *
```

```
ORGST     EQU  205000          *
*
*                    INITIALISIERUNGSLISTE
*                                DW    STARTADRESSE
*                                DW    TCB-ADRESSE
*                                DW    STACK-ADRESSE
*                                .
*                                .
*                                DW    0
*
INLIST    DW    TASK1           *
          DW    RAMANF+1000     *
          DW    RAMANF+1000     *
          DW    TASK2           *
          DW    RAMANF+2000     *
          DW    RAMANF+2000     *
          DW    TASK3           *
          DW    RAMANF+3000     *
          DW    RAMANF+3000     *
          DW    TASK4           *
          DW    RAMANF+4000     *
          DW    RAMANF+4000     *
          DW    0               *
*
*
S         EQU   RAMANF
*
TASK1     EQU   $
$$        RELEASE S
T1        EQU   $
$$        WHEN ITR1 RESUME
$$        REQUEST S
$$        PRINT 'TASK1'
$$        RUN 25 SEC
$$        RELEASE S
$$        ACTIVATE TASK2 PRIORITY 20
$$        PRINT 'END1'
          JR    T1
*
*
*
TASK2     EQU   $
$$        WHEN ITR2 ACTIVATE TASK3 PRIORITY 10
$$        WHEN ITR3 ACTIVATE TASK4 PRIORITY 5
$$        PRINT 'TASK2'
$$        SUSPEND
$$        PREVENT TASK3
$$        PRINT 'END2'
$$        TERMINATE
*
*
*
TASK3     EQU   $
$$        REQUEST S
$$        PRINT 'TASK3'
$$        RUN 15 SEC
$$        RELEASE S
$$        PRINT 'END3'
$$        TERMINATE
*
*
*
TASK4     EQU   $
$$        PRINT 'TASK4'
$$        CONTINUE TASK2
$$        TERMINATE TASK1
```