KfK 2715 B
Dezember 1978

Untersuchungen zum Realzeitverhalten von Sprachen zur Prozeßprogrammierung mit Hilfe eines Laufzeitmeßsystems

C. Uhlig
Institut für Datenverarbeitung in der Technik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Datenverarbeitung in der Technik

KfK 2715 B

Untersuchungen zum Realzeitverhalten von Sprachen zur Prozeßprogrammierung mit Hilfe eines Laufzeitmeßsystems

von

C. Uhlig

Diplomarbeit eingereicht bei der Fakultät für Informatik der Universität Karlsruhe

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
ISSN 0303-4003

Zusammenfassung

Für VARIAN-Prozeßrechner der V70-Serie wird versucht die verfügbaren Sprachen ASSEMBLER, FORTRAN, IFTRAN, PASCAL, und BASIC hinsichtlich ihrer Eignung für die Prozeßprogrammierung zu untersuchen.
Hierzu werden die Realzeitmöglichkeiten und die Effizienz dieser Sprachen zur Laufzeit verglichen.
Für die Messung von Rechenzeiten wird ein speziell entworfenes, komfortables Laufzeitmeßsystem vorgestellt, das unter dem Betriebssystem VORTEX II F der VARIAN-Rechner implementiert wurde.
Aufgrund der Ergebnisse dieser Laufzeitmessungen sollten auch höhere Sprachen zur Prozeßprogrammierung auf diesen Rechnern verwendet werden.

Examinations of the Real-Time Behaviour of Languages for Process-Programming Using a Runtime Measuring System

Abstract

The aim of this paper is to examine the languages ASSEMBLER, FORTRAN, IFTRAN, PASCAL and BASIC regarding to their suitability for process-programming.
These languages are implemented on a VARIAN computer of the V70-series and are running with the operating-system VORTEX II F.
This report compares the real-time-features and the efficiency at runtime.
Therefore an especially developed, comfortable runtime measuring system is presented.
Based on the results of execution-time measurings, high level languages should be applied to process-programming.

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation und Zielsetzung

Für die Entwicklung und Implementierung von rechnergeführten Prozeßüberwachungssystemen existieren im Rechnerversuchsfeld des Instituts für Datenverarbeitung in der Technik VARIAN-Rechner der V70-Serie.
Auf der VARIAN sind bisher keine eigentlichen Prozeßsprachen implementiert, sondern Sprachen, die für einen universellen Einsatzbereich vorgesehen sind:

ASSEMBLER, BASIC, FORTRAN, IFTRAN und PASCAL *)

Bedingt durch das Fehlen einer Prozeßsprache stellt sich die Frage, welche der verfügbaren Sprachen sich am besten für die Lösung von Aufgaben aus der Prozeßautomatisierung eignet. Eine Antwort soll durch einen Vergleich der Sprachen gegeben werden. Hierzu genügt jedoch nicht nur die Untersuchung der allgemeinen Spracheigenschaften /1/, sondern es müssen auch Aussagen über das Laufzeitverhalten der Sprache gemacht werden.

1.2 Anforderungen an eine Prozeßprogrammiersprache

Für die Prozeßprogrammierung werden neben allgemeinen Spracheigenschaften zusätzliche Elemente gefordert:

parallele Aktivitäten: Um die Dynamik des Prozesses besser anpassen zu können, wird bei der Prozeßprogrammierung die Lösung eines Problems auf mehrere selbständige Programmeinheiten (Tasks) aufgeteilt. Da diese Einheiten stark voneinander abhängen, müssen Mittel zur Verfügung stehen, um den dynamischen Ablauf zu steuern.

*) genauere Beschreibung der Sprachen siehe /1/

Interrupt-Verarbeitung:

Das Geschehen bei der Prozeßprogrammierung wird durch den überwachten Prozeß bestimmt. Signale vom Prozeß, die zu unvorhergesehenen Zeitpunkten im Rechner eintreffen (Interrupts), müssen erkannt und verarbeitet werden.

Ein/Ausgabe:

Bei der Prozeßautomatisierung fällt eine sehr intensive EA-Tätigkeit sowohl mit dem Prozeß als auch mit der Standardperipherie an. Dazu müssen von der Sprache umfassende EA-Möglichkeiten angeboten werden.

Kommunikation, Synchronisation:

Mehrere Tasks, die an der Lösung einer Aufgabe arbeiten, müssen Daten miteinander austauschen. Deshalb sind gemeinsame Datenbereiche zur Verfügung zu stellen, über welche eine Kommunikation möglich ist. Der Zugriff auf solche Datenbereiche muß aus Sicherheitsgründen über Synchronisationsmechanismen geregelt werden.

Von den untersuchten höheren Programmiersprachen bietet keine den vollen Umfang der oben geforderten Eigenschaften auf Sprachebene an. Deshalb wird zunächst aufgezeigt, wie bzw. ob die fehlenden Mechanismen über entsprechende Betriebssystemdienste angeschlossen werden können.

Daneben wird aber von einer Programmiersprache auch verlangt, daß die Ausführungszeit für den Objektcode gering ist. Gerade bei technischen Prozessen sind die Reaktionen auf gewisse Zustände der überwachten Anlage innerhalb bestimmter Zeitschranken zu erbringen, um ein richtiges Arbeiten zu ermöglichen. Deshalb muß in eine Sprachbeurteilung auch die benötigte Rechenzeit mit eingehen.

Die Belastung des Rechners durch ein Programm erfolgt aber nicht nur durch die Ausführung von Befehlen, die in einer Programmiersprache geschrieben wurden. Vielmehr sind Prozeßprogramme sehr EA-intensiv, sodaß auch der Verwaltungsaufwand des Betriebssystems (z.B. EA-Auftragsverwaltung, Treiber etc.) mit zu berücksichtigen ist. Wie sich später zeigen wird, darf gerade dieser Aufwand nicht vernachlässigt werden.
Aus diesen Gründen wurde ein System zur Laufzeitmessung entwickelt, mit dem u. a. die oben erwähnten Zeitmessungen durchgeführt werden können. Das System zeichnet sich dadurch aus, daß sehr umfangreiche Messungen möglich sind, die der Benutzer zum Teil seinen eigenen Wünschen anpassen kann. Beim Entwurf wurde darauf geachtet, daß das Laufzeitmeßsystem soweit wie möglich von Programmiersprachen unabhängig ist und der Rechner durch die Meßprogramme minimal belastet wird.

2. Das Realzeitbetriebssystem VORTEX II

In diesem Kapitel wird das Betriebssystem VORTEX II vorgestellt. Ziel dieser Beschreibung ist es, die Eigenschaften von VORTEX soweit aufzuzeigen, wie sie zum Verständnis des Sprachvergleichs und des Laufzeitmeßsystems notwendig sind. Neben einer Einführung in das allgemeine Konzept werden in den folgenden Abschnitten vor allem die Interrupt-Verarbeitung, die Ein/Ausgabe und Speicherverwaltung, sowie das Tasksteuerkonzept genauer dargelegt.

2.1 Allgemeines Konzept

VORTEX II (Varian Omnitask Real-Time-Executive) ist ein auf die Rechnerfamilie VARIAN V70 zugeschnittenes Realzeitbetriebssystem /2/.

Alle Aufgaben werden von VORTEX in Form von Tasks durchgeführt. Eine Task ist dabei die Ausführung von Befehlen unter der Regie des Betriebssystems. Sie wird gekennzeichnet durch die Attribute

- Taskname
- Taskstatus
- Priorität zur Einstufung der Dringlichkeit (wird bei der Definition angegeben und ist danach bis zum Abbruch der Task nicht mehr veränderbar)
- zugeordnete Befehlsfolge (Programm)
- logischer Adreßraum von maximal 32 K Worten zum Ablegen von Befehlen und Daten

Eine Task ist der Taskverwaltung erst dann bekannt, wenn sie in der zentralen Auftragsdatei vermerkt ist /3/. Diese Datei besteht aus der Menge aller, dem Betriebssystem bekannten und zur Bearbeitung anstehenden Aufträgen. Jede Task wird hier durch einen Taskbeschreibungsblock (TIDB Task Identification Block) repräsentiert,

in dem alle zur Ausführung notwendigen Daten wie Name, Priorität, benötigter Speicherplatz, Startadresse des zugehörigen Programms, Bereiche zum Retten von Registerinhalten usw. abgelegt sind.
Zwei Attribute sollen hier genauer beschrieben werden, da sie für die weiteren Betrachtungen notwendig sind:

- Das Interrupt Ereignis Wort (Interrupt Event Word). Falls für eine Task ein Interrupt eintrifft, wird hier ein dem Interrupt zugeordneter Wert (Bitmuster) eingetragen, genauer, mittels der Oder-Funktion in den alten Inhalt eingeblendet. Sind die Bitmuster aller Interrupts, an die eine Task angeschlossen ist, disjunkt , so kann anhand dieses Wertes eindeutig die Interruptquelle erkannt werden. Durch Eintragen von Bitmustern ist auch die Simulation von Alarmen per Software möglich /4/.
- Der Taskstatus
  Unter VORTEX sind alle Taskbeschreibungsblöcke entsprechend ihrer Priorität in einer linearen Liste miteinander verkettet. Für die verschiedenen Bearbeitungszustände einer Task (s. 2.5) werden diese Blöcke nicht in andere, den jeweiligen Taskzuständen entsprechende Listen eingetragen, sondern ihr Status wird durch das Setzen bzw. Löschen von Bits im Taskstatuswort beschrieben.

Zentrale Schaltstelle bei der Vergabe des Rechnerkerns ist der Dispatcher. Er durchläuft regelmäßig oder auf Anforderung die Kette der Taskbeschreibungsblöcke und aktiviert diejenige Task, die die höchste Priorität besitzt und auf Grund ihres gegenwärtigen Zustandes gestartet bzw. fortgesetzt werden kann.

Für die Zeitrechnung ist die Realzeituhr (RTC Real-Time Clock) zuständig. Sie unterbricht in regelmäßigen Zeitabständen (alle 5 ms) die momentan aktive Befehlsfolge zugunsten einer der Realzeituhr zugeordneten Unterbrechungsroutine. Hier werden die Systemzeit aktualisiert und bei Tasks, die für eine bestimmte Zeitdauer ausgesetzt wurden, das Ende der Verzögerungszeit überprüft. Dies wird durch Löschen der 'Delay-Flag' im Statuswort der Task gekennzeichnet, die dann wieder bei der Vergabe des Prozessors berücksichtigt wird.

VORTEX unterscheidet von seinem Konzept her 2 verschiedene Aufgabentypen, Vordergrund (Foreground) - und Hintergrund (Background) - Aufgaben.

Im Vordergrund erfolgt die Bearbeitung aller zeitkritischen Aufträge, also aller Realzeitprogramme. Ihm gegenüber steht der Hintergrund-Bereich, der sequentiell Aufgaben im Stapelbetrieb ausführt. Dazu gehören das Übersetzen und Binden von Programmen, die Dateiverwaltung und andere zeitunkritische Aufgaben. Die Zugehörigkeit einer Task zu einem dieser Bereiche wird durch Parameter beim Binden festgelegt.
Bei der Bearbeitung haben Vordergrund-Programme absoluten Vorrang vor Programmen aus dem Hintergrund. Zusätzlich bestehen aber noch eine Reihe anderer, fundamentaler Unterschiede /2/:

- Im Hintergrund kann zu einer Zeit immer nur ein Programm ausgeführt werden, während im Vordergrund die Parallelarbeit von bis zu 15 Tasks möglich ist.
- Foreground-Programme werden bei der Zuteilung des Prozessors entsprechend ihrer Prioritäten berücksichtigt. Zur Einstufung der Dringlichkeit einer Task stehen die Prioritäten 2 (niederste) bis 31 (höchste Priorität) zur Verfügung. Der Hintergrund kennt nur die Prioritäten 1 (für Systemtasks wie Compiler, Binder, Job-Control-Prozessor etc.) mit erweiterten Zugriffsrechten auf Betriebssystemkomponenten und Tabellen und Ø für Anwenderprogramme.

- Zur Steuerung der Taskausführung kann vom Vordergrund der volle Satz von Tasksteuerbefehlen benutzt werden, im Background sind nur eingeschränkte Steuermöglichkeiten vorhanden.
- Falls für Vordergrundprogramme Arbeitsspeicher benötigt wird, kann dazu das aktuelle Backgroundprogramm auf Platte ausgelagert werden. Die Auslagerung von Vordergrundprogrammen ist nicht möglich, ihnen kann eine dringlichere Task höchstens das Betriebsmittel 'MAP-KEY' entziehen (s. 2.4)
- Nur Vordergrundtasks können an Interrupts angeschlossen werden und den Taskkommunikationsbereich BLANK COMMON verwenden

2.2 Unterbrechungen

Interrupts (Unterbrechungen, Alarme) bedeuten eine Unterbrechung des momentanen Befehlsablaufs zugunsten einer dem Interrupt zugeordneten Befehlsfolge. Um auf Unterbrechungen zu antworten, stellt VORTEX 2 Möglichkeiten zur Verfügung /2/:

- Interrupt-Routinen
  Interrupt-Routinen sind kurze Programmstücke, an die entsprechend des Interrupts verzweigt wird (über Interrupt-Vektor). Am Ende dieser Routinen wird entweder die Bearbeitung des unterbrochenen Programms fortgesetzt oder der Dispatcher angesprungen, um eine andere Task zu aktivieren. Damit sind sehr schnelle Reaktionen auf Alarme möglich. Die Ausführung muß aber unter Ausschluß weiterer Unterbrechungen erfolgen, da ein erneuter Aufruf der Interrupt-Routine die vorhergehende Rücksprungadresse überschreiben würde. Deshalb dürfen hier keine EA-Aufrufe oder Tasksteuerroutinen verwendet werden, da diese die Unterbrechungssperre aufheben.
- Interrupt-Tasks
  Hier wird eine dem Interrupt zugeordnete Task aktiviert. Sie wird durch Setzen des Ereignis-Wortes vom Eintreffen des Interrupts in Kenntnis gesetzt und

anschließend entsprechend ihrer Priorität bearbeitet. Voraussetzung für den Anschluß ist, daß die Task der Taskverwaltung bekannt ist. Deshalb muß der Taskbeschreibungsblock ständig in der Auftragsdatei vorhanden sein und darf bei Ende der Task keinesfalls vernichtet werden (TIDB ist bei Systemgenerierung speicherresident zu definieren). Als Interrupt-Tasks kommen nur Foreground-Programme in Frage.

2.3 Ein/Ausgabe

Für die Kommunikation mit externen Geräten wird von VORTEX eine geräteunabhängige Schnittstelle bereitgestellt. Alle EA-Geräte werden über logische Gerätenummern angesprochen. Die Zuordnung von logischen Einheiten zu den tatsächlichen Peripheriegeräten ist flexibel und kann in weiten Teilen an vorgegebene Bedingungen angepaßt werden. Fällt z. B. ein Gerät aus, so kann die Ein/Ausgabe über einen Eingriff des Operateurs durch Ändern der Zuordnung auf eine andere, funktionsfähige Einheit umgeschaltet werden. Diese Reaktion auf Hardwarefehler darf nicht verwechselt werden, mit individuellen Reaktionen von Programmen auf fehlerhafte Datenübertragung (s. 3.2)
Der Anwender selbst darf keine direkten EA-Operationen durchführen, da dies privilegierte Befehle des Betriebssystems sind. Er erteilt lediglich der EA-Verwaltung den Auftrag, die von ihm gewünschten EA-Vorgänge auszuführen. Dafür stehen 8 elementare Operationen zur Verfügung.

READ	Lesen eines Datensatzes vom externen Gerät in den Arbeitsspeicher
WRITE	Übertragen eines Datensatzes vom Arbeitsspeicher auf ein externes Gerät
WEOF	Schreiben einer End-of-File Marke

REW	Positionierung auf den Anfang der log. Einheit
SREC	Überspringen eines Datensatzes auf der logischen Einheit (Vorwärts oder Rückwärts)
FUNC	Geräteabhängige Funktion z. B. Seitenvorschub bei Drucker Ausgabe von Leerkarten bei Kartenstanzer
OPEN	Eröffnen der Zugriffsmöglichkeit auf eine Plattendatei durch Übernahme von Plattendateiadressen in den Programmbereich
CLOSE	Beenden der Zugriffsmöglichkeit auf eine Plattendatei

Nicht auf jedes EA-Gerät sind alle oben aufgeführten Operationen anwendbar. Die Zulässigkeit der Befehle ist neben anderen, gerätespezifischen Daten in einer Controller-Tabelle enthalten, die für jedes EA-Gerät existieren muß. Diese Tabelle verfügt auch über einen Verweis auf den Anfang der Liste, in der alle EA-Aufträge für diese Einheit linear miteinander verkettet sind. Steht für ein Peripheriegerät ein Auftrag an (Zeiger≠0), so wird von der EA-Verwaltung das zugehörige Treiberprogramm gestartet, welches die Abwicklung des Datentransfers mit dem EA-Gerät überwacht /2,5/.

2.4 Speicherverwaltung

Der gesamte zur Verfügung stehende Arbeitsspeicher ist in Kacheln (Pages) von 512 Worten eingeteilt. Sie stellen die kleinste Einheit dar, die von der Speicherverwaltung vergeben werden kann.
Jede Task arbeitet in ihrem eigenen logischen Adreßraum von 32 K Worten. Mit Hilfe der Memory Map werden diese logischen Adressen mittels Hardware in physikalische Adressen umgesetzt. Dadurch ist es möglich, trotz der Wortlänge von nur 16 Bit, einen Gesamtspeicher von 256 K Worten zu adressieren.

Für die Transformation der logischen in die physikalischen Adressen werden jeder Task 64 13 bit-Register (= Anzahl der möglichen Seiten pro Task) der Memory Map zugeordnet. Diese Register können nur über einen Schlüssel (MAP-KEY) angesprochen werden, der beim Laden an die Task vergeben wird /6/.

Diese Speicherorganisation erlaubt es, Adressierungsfehler auf Hardwareebene festzustellen. Es ist deshalb nicht möglich, durch falsche Adressierung den Inhalt von Speicherbereichen zu zerstören, die anderen Tasks zugeordnet sind, da eine Task nur auf die ihr zugewiesenen Speicherplätze zugreifen kann. Das Zugriffsrecht ist anhand der Modusangabe ersichtlich, die in den Map-Registern enthalten ist.

Allen Tasks gemeinsam ist die Seite Ø (logische und physikalische Adressen Ø - 511), die globale Systemgrößen (unter anderem auch den Interrupt-Vektor) enthält. Sie ist die allgemeine Schnittstelle zwischen Benutzertasks und Betriebssystem.

VORTEX II arbeitet in 2 verschiedenen Modi:

Betriebssystem-Modus (Executive Mode)

In diesem Modus läuft das Betriebssystem sowie alle Betriebssystem-Tasks (Treiber, Ladeprogramm usw). Hier sind alle Befehle erlaubt. Dem Betriebssystem-Modus ist der MAP-KEY Ø zugeordnet, was bedeutet, daß die logischen Adressen mit den physikalischen identisch sind. Bei Unterbrechungen wird automatisch in den Executive Mode umgeschaltet.

Benutzer-Modus (User Mode)

Für den Benutzer Modus stehen die MAP-KEY's 1 - 15 zur Verfügung. Privilegierte Befehle (EA-Befehle, Steuerbefehle) sind in diesem Modus nicht zugelassen.

2.5 Taskzustandsmodell

Eine Task durchläuft während ihrer Existenz verschiedene Zustände, die ihre momentane Situation beschreiben. Diese Zustände sind erkennbar anhand der Statusworte im TIDB einer Task. Für das Betriebssystem VORTEX II lassen sich die verschiedenen Taskzustände auf folgendes Taskzustandsmodell abbilden.

Unbekannt: Eine Task befindet sich in diesem Zustand, wenn sie der Taskverwaltung nicht bekannt ist. Sie kann jedoch schon als Lademodul, etwa auf Platte, existieren.

Definiert: Ist der Auftrag der Taskverwaltung bekannt, d. h. existiert ein Taskbeschreibungsblock, so gilt die Task als definiert. In diesem Zustand sind die notwendigen Betriebsmittel Speicherplatz und MAP-KEY noch nicht zugeteilt oder das Programm befindet sich in nicht ausführbarer Form im Arbeitsspeicher.

Bereit: Die Task wartet auf die Zuteilung des Prozessors. Alle übrigen Betriebsmittel wie Speicherplatz und MAP-KEY sind vorhanden. Das zugehörige Programm befindet sich im Arbeitsspeicher.

Rechnend: Eine Task befindet sich im Zustand rechnend, wenn sie über den Prozessor verfügt, d. h. wenn sie Befehle ausführt.

Suspendiert: Eine Task ist suspendiert, wenn sie auf das Eintreffen eines bestimmten Ereignisses wartet. Dieses Ereignis kann ein Interrupt sein, der Fortsetzungsbefehl durch eine andere Task (RESUME), das Ende einer Task oder das Ende einer E/A (falls darauf gewartet wird).

Verzögert: Eine Task befindet sich im Zustand verzögert, wenn sie auf den Ablauf einer bestimmten Zeitspanne wartet.

Beendet: Eine Task ist beendet, wenn sie durch das Setzen der 'Ende-Flag' im Statuswort signalisiert, daß sie aus der Auftragsdatei entfernt werden kann.

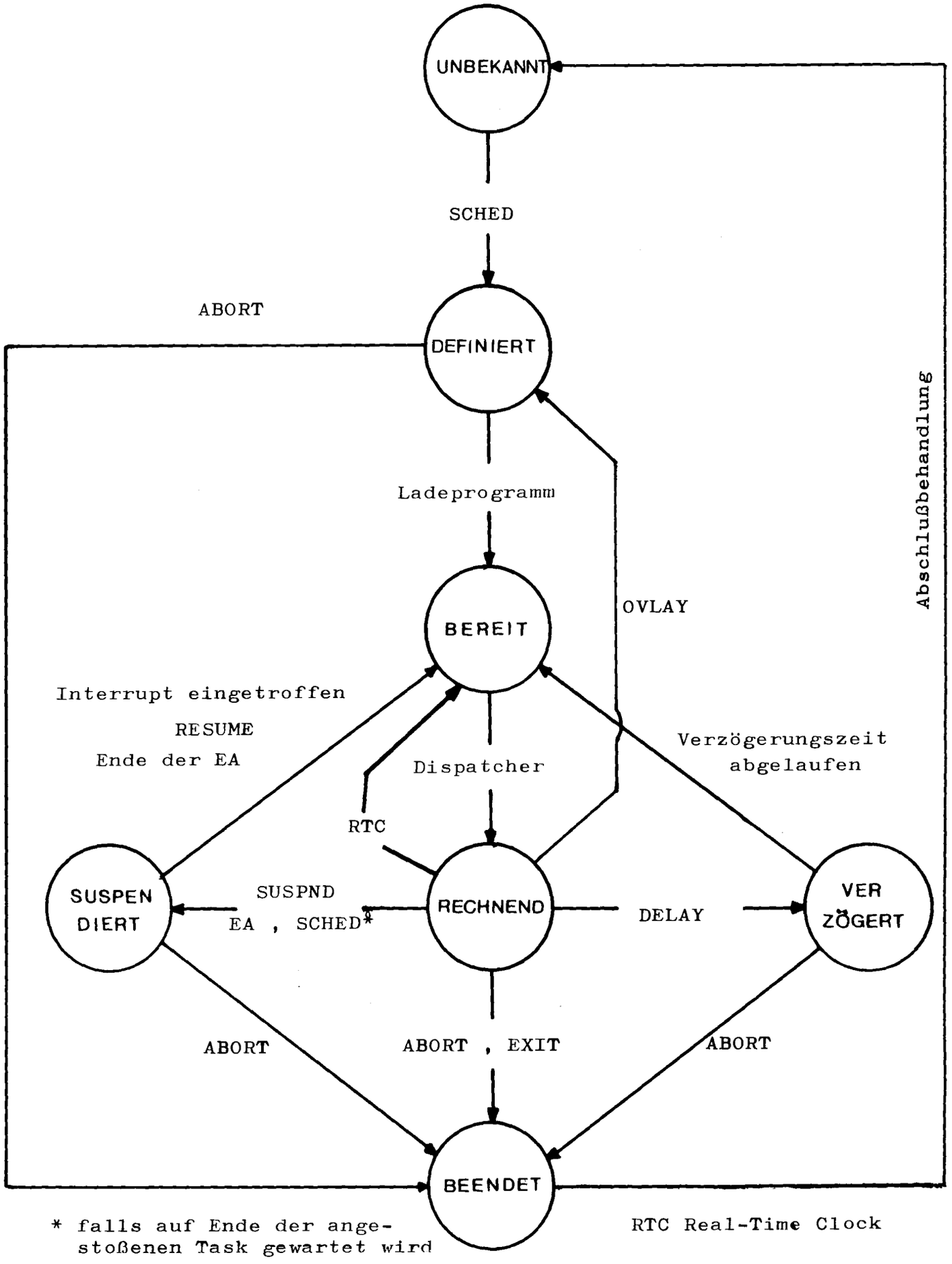


Bild 2.1 Taskzustände und Übergänge im Betriebssystem VORTEX II

2.6 Tasksteuerung

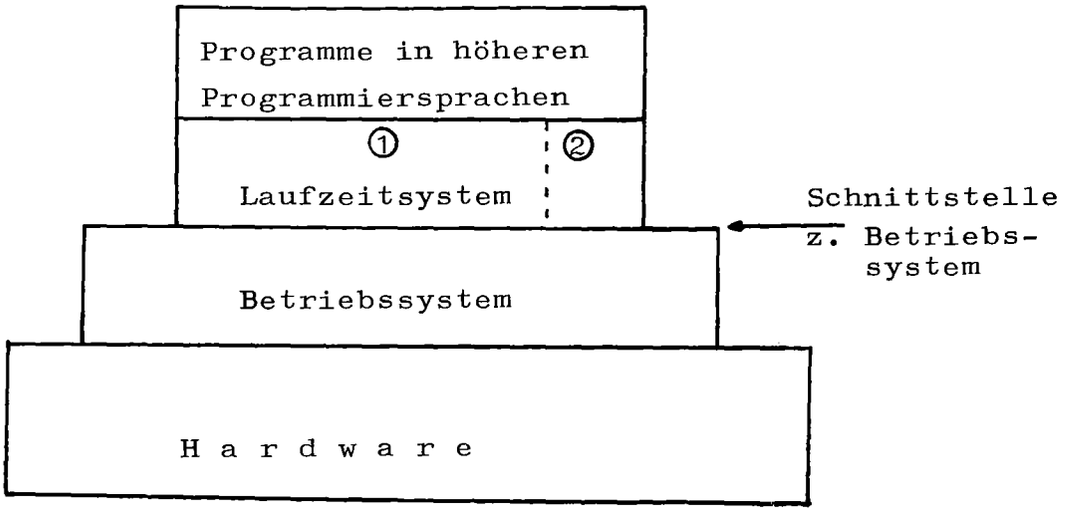
Über von VORTEX angebotene Systemdienste kann ein Zustandswechsel in definierter Weise vollzogen werden. Da die Tasksteuerung ein Kennzeichen der Realzeitprogrammierung ist, sind die Aufrufe zum Zustandswechsel nur von Foreground-Tasks in vollem Umfang verwendbar. Diese Befehle sind im einzelnen /2/:
Über das SCHEDULE-Kommando wird ein Taskbeschreibungsblock für die Task angelegt und in die Auftragsdatei eingefügt. Nur hier besteht die Möglichkeit, die Priorität der Task zu definieren.
Der Übergang in den Zustand 'bereit' ist nur vom Betriebssystem über das Ladeprogramm vollziehbar, das einer Task die Betriebsmittel Speicher und MAP-KEY zuweist.
In den Zustand 'rechnend' gelangt eine Task nur über den Dispatcher, falls sie von allen Tasks, die bereit sind, die höchste Priorität besitzt.
Über den SUSPEND-Befehl kann sich die Task selbst von der weiteren Bearbeitung ausschließen. Die Fortsetzung ist dabei an das Eintreten eines bestimmten Ereignisses geknüpft.
Mit dem RESUME-Befehl kann die Suspendierung einer anderen Task aufgehoben werden.
Das Aussetzen für eine bestimmte Zeitdauer ist über DELAY möglich. Nach Ablauf der Verzögerungszeit geht die Task in den Zustand bereit über.
Durch das OVERLAY-Kommando können Overlay-Segmente eingelagert werden. Die Task wechselt in den Status definiert, da sich das Programm nicht vollständig im Arbeitsspeicher befindet.
Über EXIT bzw. ABORT kann die Bearbeitung von Tasks eingestellt werden. Bei nächster Gelegenheit wird der TIDB dieser Tasks aus der Auftragsdatei entfernt und die belegten Betriebsmittel werden freigegeben.

Regelmäßig mit einem RTC-Interrupt geht die gerade rechnende Task in den Bereitzustand zurück. Am Ende der Interrupt-Routine wird an den Dispatcher verzweigt, der anschließend den Prozessor an die Task mit der höchsten Priorität zuweist.

3. Wechselwirkung zwischen Betriebssystem und Programmiersprache

VORTEX stellt für die Tasksteuerung, die Ein/Ausgabe und andere Funktionen eine Reihe von Hilfsprogrammen zur Verfügung, die über eine fest-definierte Assembler-Schnittstelle aufgerufen werden.

Um von einem Betriebssystem unabhängig zu sein, enthalten höhere Programmiersprachen meist nur einige dieser Dienstleistungen als Sprachmittel. Nicht darstellbare Systemfunktionen müssen über externe Unterprogramme angeschlossen werden, die eine Erweiterung des Laufzeitsystems bilden.



1 Anschluß von Dienstleistungen, die als Sprachmittel vorhanden sind
2 Anschluß von Betriebssystemfunktionen, die nicht durch die Sprache beschrieben werden können

Bild 3.1 Programmschichten

Die Lösung von Realzeitproblemen sieht fast immer die Aufgabenteilung auf mehrere Tasks vor. Das bedeutet, daß zu der intensiven Ein-Ausgabe mit dem Prozeß auch eine starke Kommunikation zwischen Task und Betriebssystem hinsichtlich Tasksteuerung hinzukommt. Ursprünglich nicht für die Programmierung von Realzeitaufgaben entwickelt, stellt keine der untersuchten höheren Sprachen die geforderten Funktionen in vollem Umfang zur Verfügung; selbst das Betriebssystem VORTEX II bietet nicht alle notwendigen Leistungen an. Die folgenden Abschnitte zeigen auf, welche dieser Funktionen von den Programmiersprachen zur Verfügung gestellt werden, bzw. welche über externe Unterprogramme angesprochen werden können.

3.1 Real-Time-Executive Routinen (RTE-Routinen)

Unter der Betriebssystemkomponente EXEC werden dem Anwender Mittel zur Tasksteuerung, Aufrufe zur Speicherplatzverwaltung und andere Hilfsdienste angeboten. Für den folgenden Vergleich werden nur diejenigen Service-Routinen herangezogen, die für die Realzeitprogrammierung von Interesse sind

a) Tasksteuerung
   SCHED, SUSPND, DELAY, RESUME, EXIT, ABORT
   mit den Bedeutungen aus 2.6

b) Speicherplatzverwaltung
   MAPIN Definition von gemeinsam benutztem Speicherraum (s. 3.5)
   OVLAY Laden von OVERLAY-Segmenten

c) Hilfsroutinen
   ALOC Aufruf von Reentrant-Programmen
   PMSK Setzen bzw. Öffnen von Interrupt-Masken
   TIME Übernehmen der Systemzeit
   PASS Auslesen von Systemzellen z. B. Datum, TIDB-Informationen

TBEVNT Abfragen, Setzen bzw. Löschen des Interrupt-Event-Words

In Tabelle 3.1 sind die Aufrufformen der untersuchten Programmiersprachen für diese Dienstleistungen gegenübergestellt.
Aus dem Vergleich geht hervor, daß mit Ausnahme von BASIC alle anderen Sprachen über die notwendigen Mechanismen verfügen bzw. diese über externe Unterprogramme zugänglich sind.

	ASS.	FORTRAN	IFTRAN	PASCAL	BASIC
SCHED	1	2	2	3	3**
SUSPND	1	2	2	3	3
RESUME	1	2**	2**	3	4
DELAY	1	2	2	3	4
EXIT	1	1*	1*	1*	1*
ABORT	1	2	2	3	4
OVLAY	1	2	2	4	4
MAPIN	1	4	4	4	4
PMSK	1	2	2	3	4
TIME	1	2	2	3	3
ALOC	1	2	2	3	4
TBEVNT	1	3	3	3	3
PASS	1	2	2	3	3

1 Aufruf als Sprachmittel vorhanden
2 Aufruf über Unterprogramm (in Bibliothek vorhanden)
3 Aufruf über Unterprogramm (nicht in Bibliothek vorhanden)
4 nicht möglich
** siehe Text
* jeweils gegeben, wenn Ende des Programms erreicht ist
FORTRAN/IFTRAN: STOP bzw. RETURN im Hauptprog.
PASCAL: Wenn letztes END im Hauptprog. erreicht ist (END.)
BASIC: END - Befehl

Tabelle 3.1 RT-Möglichkeiten der untersuchten Sprachen

Der auf der VARIAN V 75 implementierte BASIC-Interpreter läuft nur als Hintergrundprogramm. Für solche Programme sind von VORTEX nur eingeschränkte RTE-Aufrufe zugelassen. Beim Aufruf von bestimmten Routinen ist darauf zu achten, daß die Parameter als Bitmuster bzw. Integerzahl interpretiert werden. Da BASIC nur Zahlen in REAL-Darstellung kennt, dürfen Parameter nicht das Ergebnis von komplexeren Rechenoperationen sein (höchstens Addition bzw. Subtraktion, eingeschränkt Multiplikation); durch die Umwandlung von Real nach Integer können Informationsverluste auftreten, die eine richtige Verwendung der Service-Routinen nicht mehr gewährleisten.

Der Aufruf des SCHED-Kommandos ist beschränkt auf das Starten von Background-Tasks, die jedoch nicht zeitlich parallel abgearbeitet werden können.

FORTRAN wird vom Betriebssystem unterstützt, deshalb sind viele Dienstleistungen als Unterprogramme in einer Systembibliothek enthalten, die direkt von FORTRAN/IFTRAN benutzt werden kann. Im existierenden FORTRAN-System ist das PAUSE-Statement identisch mit einer bestimmten SUSPEND-Version (Fortsetzung der Task über RESUME). Der momentane PASCAL-Interpreter (sequential PASCAL) erlaubt nicht das parallele Arbeiten mehrerer in PASCAL geschriebener Tasks, da sie sich gegenseitig den Speicherplatz wegnehmen.

3.2 Ein/Ausgabe-Möglichkeiten der Sprachen

Ein Kennzeichen von Prozeßprogrammen ist eine sehr intensive Kommunikation mit der Umwelt. Es müssen z. B. Prozeßdaten erfaßt, auf Platte abgelegt und alphanumerisch oder graphisch auf einem Sichtgerät dargestellt werden.

Der Datenaustausch erfolgt einerseits über Standardperipherie wie Drucker, Magnetband, Platte usw. andererseits wird die Verbindung zum technischen Prozeß über die Prozeß-EA vollzogen.

Da prinzipiell alle vom Betriebssystem bereitgestellten EA-Möglichkeiten über externe Assembler-Unterprogramme an alle höheren Programmiersprachen angeschlossen werden können, soll im folgenden nur untersucht werden, welche EA-Möglichkeiten als Sprachmittel vorhanden sind.
Es wird nur die Kommunikation mit den Standard-Peripheriegeräten betrachtet. Die Prozeß EA mit dem vorhandenen Prozeßinterface wird in /1/ aufgezeigt.

Alle vom Betriebssystem angebotenen elementaren EA-Operationen (s. 2.3) können vom Assembler über Makro-Aufrufe benutzt werden. Die Steuerparameter für die verschiedenen Modifikationen der einzelnen Befehle sind voll zugänglich; dafür ist kein zusätzlicher Komfort wie Zahlkonvertierung vorhanden.

Bei FORTRAN/IFTRAN kann die Ein-Ausgabe mit oder ohne Formatierung durchgeführt werden; die Beschreibung des Formats ist über das FORMAT-Statement möglich, das eine große Zahl von Konvertierungsroutinen anbietet. Zusätzlich zur EA mit externen Geräten sind über ENCODE/DECODE auch formatiertes Schreiben bzw. Lesen zwischen programminternen Puffern möglich /7/.

PASCAL kennt eine wortweise (READ/WRITE) und eine blockweise (GET/PUT) Ein-Ausgabe. Der wortweise Datentransfer (ASCII-Darstellung) erfolgt von bzw. in einem programminternen Puffer. Beim Schreiben wird der Pufferinhalt erst dann zum Peripheriegerät übertragen, wenn er gefüllt ist oder die Ausgabe explizit verlangt wird.

	ASS	FORTRAN	IFTRAN	PASCAL	BASIC
READ	1	1	1	1	1
WRITE	1	1	1	1	1
WEOF	1	1	1	3	3
REW	1	1	1	3	3
SREC	1	1*	1*	3	3
FUNC	1	1**	1**	3	3
OPEN	1	2	2	1	3
CLOSE	1	2	2	1	3

1 Aufruf als Sprachmittel vorhanden
2 Aufruf über Bibliotheksunterprogramme
3 Aufruf über Unterprogramme möglich

* nur rückwärts über BACKSPACE
** über FORMAT-Anweisung

Tabelle 3.2 EA-Möglichkeiten der untersuchten Sprachen

Diese Puffer müssen für jedes Gerät, mit dem Daten ausgetauscht werden sollen, durch OPEN bzw CLOSE eingerichtet bzw. abgeschlossen werden. Zur Beschreibung des EA-Formats und zur Zahlkonvertierung existieren in PASCAL keine Hilfsmittel.

BASIC besitzt nur eine einfache EA-Möglichkeit auf Operateurkonsole (Schreiben bzw. Lesen), die generell eine Konvertierung beinhaltet.

Bei den EA-Aufrufen der höheren Programmiersprachen wird standardmäßig auf das Ende des Transfervorgangs gewartet. Es besteht nicht die Möglichkeit, die Programmabarbeitung parallel zur Ein-Ausgabe fortzusetzen.

Neben der Initiierung eines EA-Vorgangs ist es bei der Prozeßprogrammierung wichtig, über Statusmeldungen den erfolgreichen Abschluß der Übertragung zu überprüfen. Tritt während der Datenübertragung ein Fehler auf, so muß eine individuelle Reaktion vom Programm aus möglich sein, z. B. die Übertragung zu wiederholen oder eine Meldung auszugeben.

Die Statusinformation ist in den durch den EA-Aufruf angelegten Datenblöcken enthalten. Bei den höheren Programmiersprachen ist diese Datenstruktur unsichtbar, sodaß der Programmierer nicht direkt darauf zugreifen kann.

In Assembler kann der EA-Status direkt ausgewertet werden. Über die Service-Routine STAT ist es möglich, das Programm abhängig von der Statusinformation zu verzweigen.

STAT Adr. des EA-Befehls, err, aaa, bbb, busy

err = Sprungadresse bei Fehler
aaa = Sprungadresse bei End of File oder Anfang des Gerätes
bbb = Sprungadresse bei Geräteende
busy = Sprungadresse, falls EA noch nicht abgeschlossen.

Bei FORTRAN/IFTRAN ist der Status des letzten EA-Vorgangs über das Unterprogramm

```
CALL IOCHK (Status)
```

zugänglich /5/. Die Integer-Variable 'Status' kann folgende Werte besitzen:

-1 E/A - Fehler
Ø Normale Durchführung
1 Datei- oder Geräteende

Zusätzlich können READ/WRITE Anweisungen mit Fehlerausgängen benutzt werden.

READ bzw. WRITE (Gerät, Format, END = Endemarke,
ERR = Errormarke)

Bei fehlerhaftem Datentransfer wird das Programm an der Errormarke fortgesetzt, bei End-of-File oder End-of-Data bei der Endemarke. Eine Unterscheidung, welcher spezielle Fehler vorliegt, ist nicht möglich.

Bei BASIC stehen keine Statusangaben über die EA-Ausführung zur Verfügung, in PASCAL kann bei blockweiser EA die Anzahl der tatsächlich übertragenen Worte abgefragt werden.

3.3 Reentrant-Programmierung

Bei Reenttrant-Programmen ist eine strikte Trennung zwischen Daten und Befehlen notwendig, da diese Programmstücke zeitlich parallel von mehreren Tasks aufgerufen werden können. Das VORTEX-Betriebssystem sieht zur Ablage der lokalen Daten einen Keller vor. Bei jedem Aufruf eines Reentrant-Programms wird durch Erhöhen des Kellerzeigers eine bestimmte Anzahl von Worten reserviert. Diese Realisierung läßt keine EA-Aufrufe oder RTE-Service-Routinen zu, da sie die Organisation des Kellers zerstören würden (der für jeden Aufruf angelegte Datenbereich kann durch den programmierten Taskwechsel vertauscht werden). Der von VORTEX vorgeschriebene Aufbau von Reentrant-Programmen kann nur in Assemblerprogrammen realisiert werden. Bei höheren Programmiersprachen hat der Benutzer keine Kontrolle über den erzeugten Code. Einerseits ist es nicht möglich, den standardisierten Aufbau zu erreichen, andererseits

weiß der Benutzer nicht, ob der Compiler lokale Variable (z. B. Schleifenzähler etc) implizit anlegt.

3.4 Reaktion auf Interrupts

Der Anwender kann über 2 Wege auf Interrupts reagieren:

- Anschluß über Interrupt-Routinen
- Anschluß über Interrupt-Tasks

Da zum Retten von Registern und zum Sicherstellen des MAP-Status der unterbrochenen Task Hilfsprogramme zur Verfügung stehen, können Unterbrechungsroutinen in **ASSEMBLER**, FORTRAN und IFTRAN geschrieben werden. PASCAL steuert seine Speicherverwaltung über RTE-Kommandos, kann hier also nicht eingesetzt werden (s. 2.2).
Interrupt-Tasks werden durch eine Meldung im Taskbeschreibungsblock über das Auftreten eines Interrupts informiert und vom Dispatcher entsprechend ihrer Priorität gestartet. Zum Schreiben von Interrupt-Tasks sind keine besonderen Vorschriften zu beachten, jede Foreground-Task ist als Interrupt-Task einsetzbar. Deshalb können alle Programmiersprachen mit Ausnahme von BASIC (nur Background) verwendet werden.

3.5 Kommunikation, Synchronisation

In Anlehnung an FORTRAN stellt VORTEX den BLANK COMMON als gemeinsamen Datenbereich zur Verfügung, der allen Foreground-Tasks zugänglich ist. Er wird zu allen Programmen dazugebunden, die Daten in ihm definieren. Dies ist jedoch nur von ASSEMBLER, FORTRAN und IFTRAN aus möglich. In PASCAL und BASIC kann diese Definition nicht erfolgen, da der Compiler bzw. Interpreter den BLANK COMMON als Datenbereich nicht kennt. Der Anschluß kann jedoch über Assembler-Unterprogramme vollzogen werden, die Daten in diesem Bereich anlegen. Hierzu muß allerdings bemerkt werden, daß auch das Verändern, bzw. Lesen von Daten im BLANK COMMON über externe Unterprogramme erfolgen muß, während bei ASSEMBLER, FORTRAN und

IFTRAN die Daten unmittelbar angesprochen werden.

Eine weitere Möglichkeit, gemeinsame Datenbestände zwischen mehreren Tasks aufzubauen, wird über das MAPIN-Kommando angeboten.

Wie in Kapital 2.4 beschrieben, arbeitet jede Task in ihrem eigenen logischen Adreßraum. Die Abbildung der logischen auf die physikalischen Adressen erfolgt durch die Memory Map. Das MAPIN-Kommando ordnet nun den logischen Adressen zweier Tasks die gleichen physikalischen Adressen zu. Die gemeinsamen Datenbereiche werden dabei nur zwischen bestimmten Tasks eingerichtet, während auf den BLANK COMMON alle Foreground-Tasks zugreifen können.

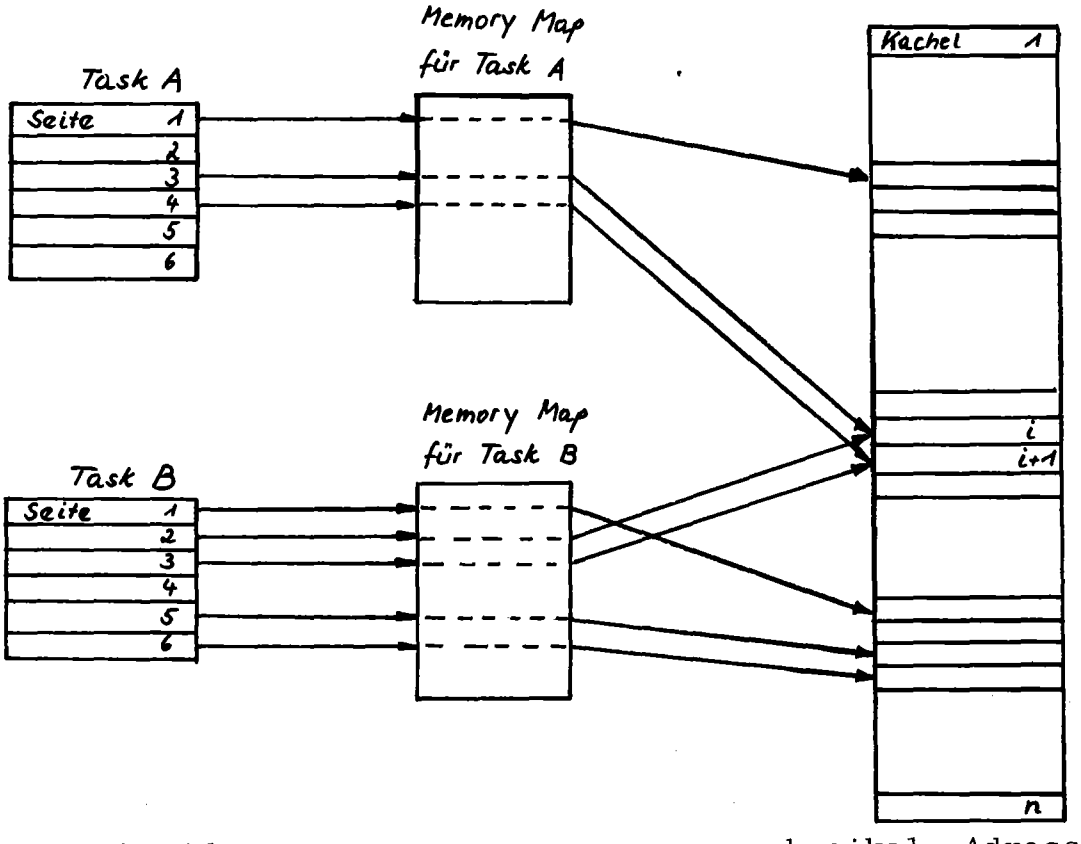


Bild 3.2 Benutzung gemeinsamer Speicherbereiche

Eine weitere Kommunikation ist durch Botschaftensysteme (send-receive Mechanismen) möglich. Diese sind aber wegen ihrer längeren Ausführungszeit nur bedingt für die Realzeitprogrammierung geeignet und werden von der gegenwärtigen Betriebssystem-Version nicht bereitgestellt.

Die Existenz von gemeinsam benutzten Daten erfordert natürlich auch genau definierte Zugriffsrechte. Eine Task darf keine Daten verändern, auf die eine andere noch zugreift. Zur Synchronisation bieten sich hier besonders Semaphore an, die genau regeln, wann gemeinsame Daten verändert werden dürfen. Die Benutzung von Semaphoren wird von VORTEX nicht unterstützt, auch in den Sprachentwürfen sind keine Sprachmittel für diesen Zweck enthalten. Daraus ergibt sich bei Verwendung von gemeinsamen Datenbereichen im gegenwärtigen VORTEX-System eine große Unsicherheit, die nur vom Benutzer durch sehr sorgfältige Programmierung vermieden werden kann.

3.6 Vergleich der Sprachen hinsichtlich der Schnittstelle zum Betriebssystem

Für die verschiedenen Programmiersprachen ergibt sich bei zusammenfassender Betrachtung folgendes Bild

	ASS.	FORTRAN	IFTRAN	PASCAL	BASIC
Tasksteuerung	x	x	x	x	-
Hilfsroutinen	x	x	x	x	-
Ein-Ausgabe	x	x	x	x	x
Statusabfrage	x	x	x	-	-
Reentrant-Prog	x	-	-	-	-
Interpt.-Rout.	x	x	x	-	-
Interpt.-Task	x	x	x	x	-
Kommunikation	x	*	*	-	-
Synchronisat.	-	-	-	-	-

x vorhanden
* teilweise vorhanden
\- nicht vorhanden

Tabelle 3.3 Vergleich der Realzeitmöglichkeiten

Mit Ausnahme von BASIC, das hier nur im Background läuft und damit nicht für die Realzeitprogrammierung verwendet werden kann, verfügen alle anderen Sprachen über die wichtigen Mechanismen bzw. diese lassen sich über externe Unterprogrammen anschließen.

Dabei ist jedoch zu berücksichtigen, daß bei FORTRAN, IFTRAN und PASCAL viele Betriebssystemfunktionen erst über eine Assembler-Schnittstelle zugänglich sind. Da jedoch keine dieser Sprachen für die Realzeitprogrammierung mit ihren besonderen Eigenheiten konzipiert wurde, wäre ein Vergleich nur aufgrund vorhandener Sprachmittel zur Beschreibung solcher Dienstleistungen unrealistisch. An dieser Stelle sei an spezielle Prozeßprogrammiersprachen, wie etwa PEARL, verwiesen, die über einen kompletten Satz von Tasksteuerbefehlen verfügen.

Reentrant-Programme bzw. Interrupt-Routinen können mit den höheren Sprachen nur teilweise oder überhaupt nicht geschrieben werden. Diese Tatsache ist jedoch nicht überzubewerten, da diese Programmtypen eigentlich Spezialfälle darstellen. VORTEX verlangt, daß solche Programme ins Betriebssystem integriert werden müssen. In dieser Umgebung herrschen andere Voraussetzungen (privilegierte Befehle, volle Zugriffsrechte auf alle Systemtabellen) die durch höhere Programmiersprachen überhaupt nicht ausgenutzt werden könnten, was ihre Effektivität beschränken würde. Aus diesem Grunde sollte für solche Probleme bei VORTEX generell der ASSEMBLER vorgezogen werden.

Für ASSEMBLER, FORTRAN und IFTRAN steht als Kommunikationsbereich der BLANK COMMON zur Verfügung; darüberhinaus können bei ASSEMBLER zusätzlich gemeinsame Daten zwischen individuellen Tasks aufgebaut werden. Bei PASCAL besteht keine Kommunikationsmöglichkeit mit anderen Tasks bzw. nur auf eine sehr umständliche Weise; außerdem ist keine Statusabfrage nach EA-Vorgängen vorhanden.

4. System zur Laufzeitmessung

Will man eine Sprache beurteilen, ob sie für den Einsatz zur Prozeßprogrammierung geeignet ist, so darf dies nicht nur hinsichtlich der Sprachmittel erfolgen, die die Sprache bereitstellt. Es ist auch wichtig, in welchem Maße damit geschriebene Programme den Rechner belasten. Um Daten über das Laufzeitverhalten der untersuchten Sprachen zu erhalten wurde, für das Betriebssystem VORTEX II F ein System zur Laufzeitmessung, kurz MONITOR genannt, entworfen und implementiert.

4.1 Entwurfskriterien, Randbedingungen

Um Leistungsdaten über einen sich in Betrieb befindlichen Rechner zu gewinnen, kann man sich zweier Methoden bedienen /8/:

- Anschluß eines Hardware-Monitors
- Anschluß eines Software-Monitors

Hardware-Monitore sind elektronische Meßapparaturen, die Aktivitäten und Zustände des Rechners in Form von elektrischen Signalen aufnehmen und abspeichern. Um Messungen durchführen zu können, müssen in den verschiedenen Hardwareeinheiten des Rechners Anschlußpunkte gesetzt werden.

Der Vorteil solcher Meßverfahren ist darin zu sehen, daß das Einsammeln der Daten parallel zur Arbeit des Rechners erfolgt, d. h. das Meßobjekt wird in seiner Arbeit überhaupt nicht beeinflußt. Mit solchen Apparaturen kann man aber nur feststellen, daß etwas geschieht, aber nicht, von wem diese Aktivität ausgeht.

Diese Meßmethode war für das vorliegende Problem nicht geeignet. Deshalb wurde das Laufzeitmeßsystem nach dem zweiten Verfahren, also als Software-Monitor entwickelt.

Bei diesen Meßverfahren werden Daten und Zustandsgrößen über programmierte Meßpunkte mit Hilfe zugehöriger Meß-

programme aufgenommen. Damit sind beliebige Messungen durchführbar. Für die Erstellung von Software-Monitoren sind aber einige Einschränkungen zu berücksichtigen /8/

- Die Meßprogramme können, falls nicht ein zweiter Rechner zur Verfügung steht, nicht parallel ausgeführt werden, sondern verbrauchen Zeit und belegen Betriebsmittel. Damit wird das zu messende Objekt belastet und die Ergebnisse verfälscht.
- Um effektiv arbeiten zu können, sind Software-Monitore ausgelegt auf ein bestimmtes Betriebssystem, bzw. eine bestimmte Betriebssystemversion. Bei Änderung dieses Systems ist deshalb auch der Monitor zu überarbeiten.
- Um die Meßprogramme richtig zu installieren, ist eine detaillierte Kenntnis des Betriebssystems notwendig.

Für die Entwicklung des Laufzeitmeßsystems waren folgende Kriterien ausschlaggebend:

- Es sollen möglichst umfangreiche Informationen über die Ausführung von Programmen gesammelt werden.
- Der Monitor sollte unabhängig von der zu untersuchenden Programmiersprache sein. Die Programme sollen in ihrem ursprünglichen Zustand belassen und nicht mit Aufrufen an Meßprogramme instrumentiert werden.
- Die Meßpunkte im Betriebssystem sollen dynamisch auf- bzw. abgebaut werden können, d. h. die Meßstellen werden nur für die Zeitspanne generiert, in der der Benutzer die Überwachung seiner Programme wünscht.
- Während der Überwachungszeit soll der normale Rechenbetrieb weiterlaufen; die Tätigkeit des Monitors soll nach außen nicht sichtbar sein.
- Der Monitor soll leicht erweiterbar sein, um an neue Gegebenheiten (neue EA-Geräte, spezielle Wünsche des Benutzers) angepaßt werden zu können.

4.2 Verschiedene Ebenen der Messung

Will man mit Hilfe von Laufzeitmessungen Vergleiche zwischen Programmiersprachen anstellen, so genügt hierzu nicht allein das Messen der Rechenzeit, die das Programm benötigt. Es muß streng unterschieden werden zwischen der Zeit, die für die Ausführung der in der Programmiersprache geschriebenen Befehle verbraucht wird und dem Zeitaufwand für das Ausführen von Dienstleistungsroutinen des Betriebssystems. Auch für die Taskverwaltung des Betriebssystems ist ein gewisser Aufwand erforderlich, der gerade bei Prozeßprogrammen mit mehreren Tasks nicht unterschätzt werden darf. Der dazu entworfene Monitor trägt diesen Gegebenheiten Rechnung, da mit ihm Messungen auf 3 Ebenen vorgenommen werden können.

- Betriebssystemebene

  Hier wird der gesamte Verwaltungsaufwand gemessen, der während der Überwachungszeit anfällt (Treiber, Ladeprogramm, Interrupt-Routinen etc). Gesondert wird die Zeit gemessen, die der Dispatcher zum Aktivieren von Tasks benötigt. Nur auf dieser Ebene kann festgestellt werden, wie lange der Prozessor während der Überwachungsphase nicht aktiv war.

- Taskebene

  Das Ergebnis der Laufzeitmessungen auf dieser Stufe ist unmittelbar abhängig von der verwendeten Programmiersprache. Aus diesem Grund werden hier die detailliertesten Messungen vorgenommen. Es wird neben der Ausführungszeit für die Befehle der jeweiligen Programmiersprache auch die Zeiten für den Aufruf von Betriebssystemroutinen, sowie die Zeiten gemessen, die die Task in den einzelnen Taskzuständen verbringt. Diese Daten werden getrennt für jede Task akkumuliert.

- Programmebene

  Auf Taskebene ist es unmöglich, Zeitmessungen für Programmteile (Unterprogramme, Schleifen etc) getrennt zu messen, da hier das Programm als eine Einheit betrachtet wird und eine Gliederung in einzelne Komponenten nicht sichtbar ist. Nur der Benutzer kann bestimmen, welche Teile überwacht werden sollen; so läßt es sich in diesem Fall nicht umgehen, das Programm mit dem Aufruf von Meßprogrammen zu instrumentieren. Es wurde jedoch eine Methode entwickelt, die diese Instrumentierung sehr einfach gestaltet.

Für die Hierarchie der 3 Ebenen ergibt sich folgender Aufbau:

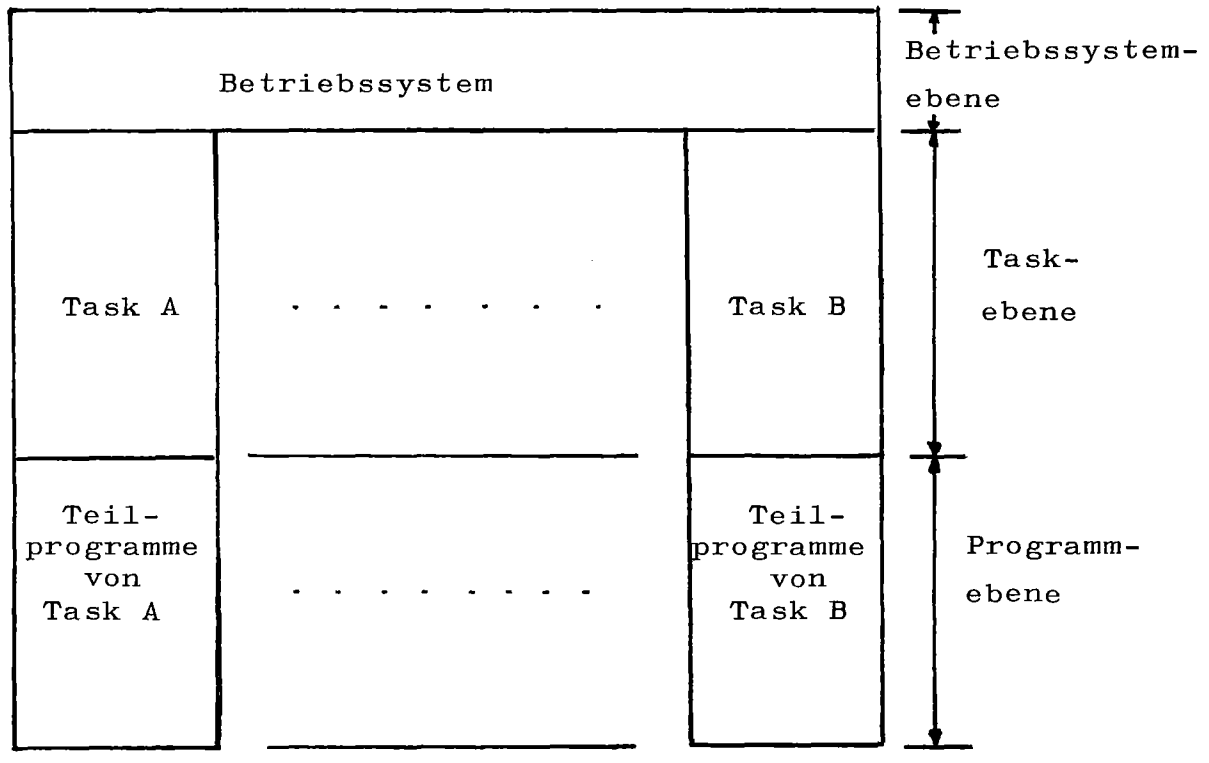


Bild 4.1 Hierarchie der 3 Meßebenen

Grundlegende Unterschiede zwischen den einzelnen Stufen sind in der folgenden Tabelle aufgeführt:

Betriebssystemebene Taskebene	Programmebene
Meßprogramme sprachunabhängig	Meßprogramme sprachabhängig
zu überwachende Programme müssen nicht instrumentiert werden	Programme müssen instrumentiert werden
Meßpunkte im Betriebssystem müssen vorhanden sein	Nur Clock-Routine muß zur Aktualisierung der Monitorzeit instrumentiert sein
Tabellen zur Ablage der Meßdaten werden im Betriebssystembereich verwaltet	Speicherbereiche zur Aufnahme der Meßdaten müssen im Programm angelegt werden

Tabelle 4.1 Unterschiede zwischen Betriebssystem-/Task-Ebene und Programmebene

Das Zusammenspiel zwischen Betriebssystem-/Task-Ebene und Programmebene ist im Bild auf der nächsten Seite dargestellt.

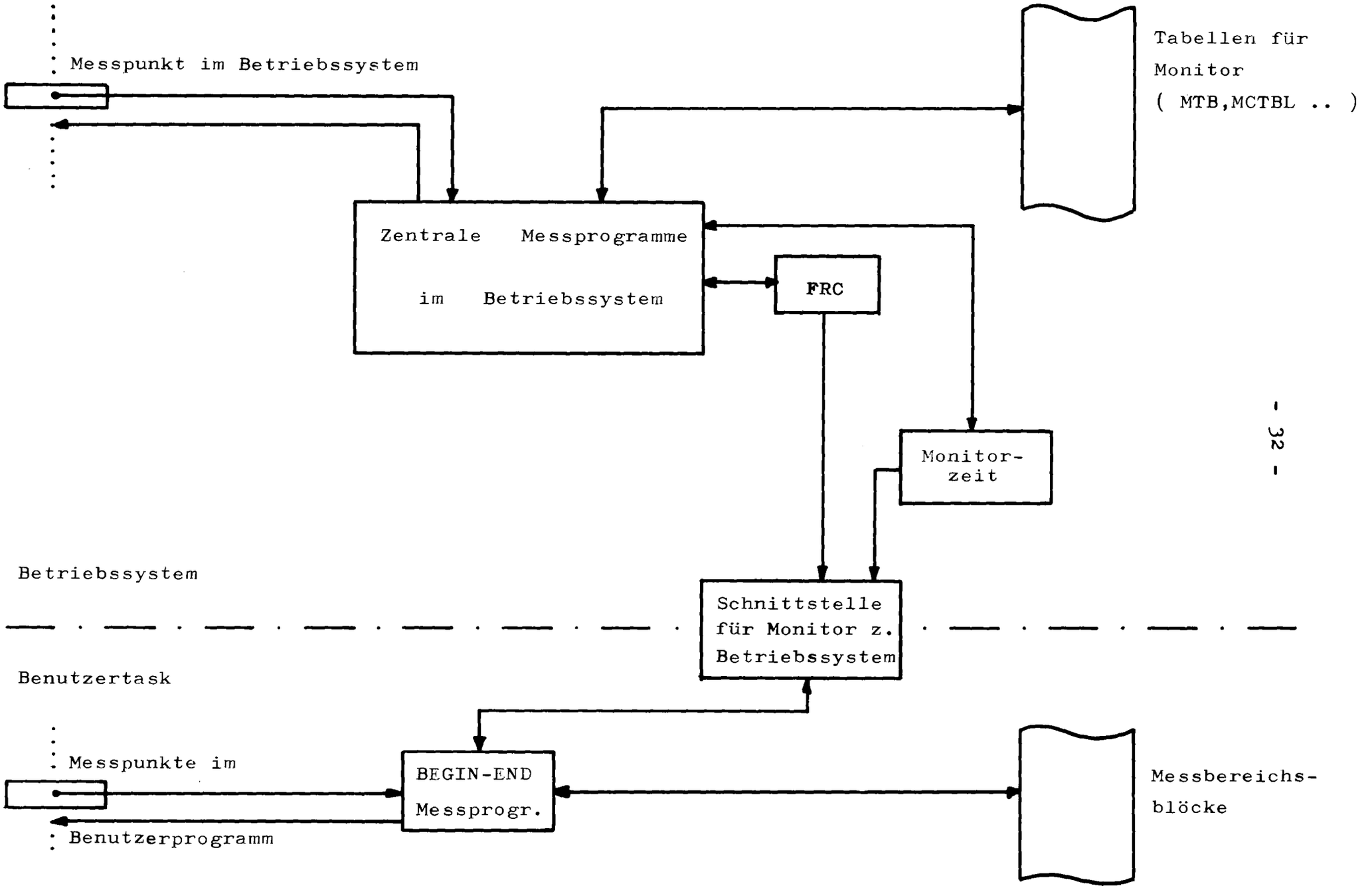


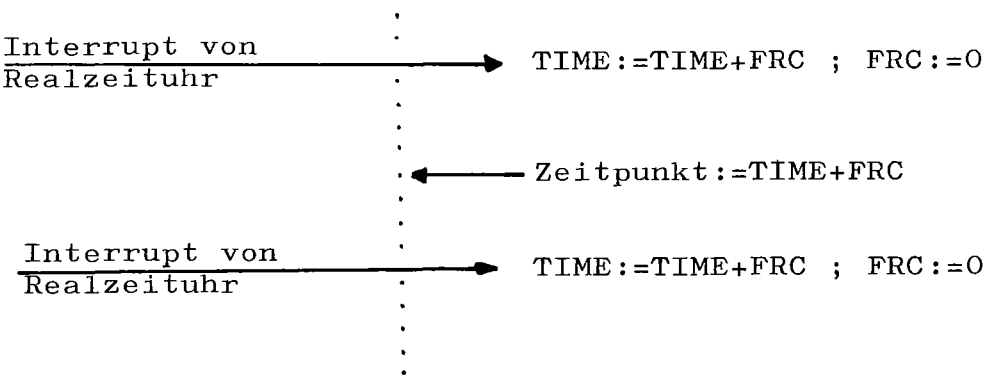
Bild 4.2 Ablauf der Messungen FRC Free Running Counter

4.3 Monitorzeit

Für Zeitmessungen steht nur die Realzeituhr zur Verfügung. Diese liefert höchstens alle 5 ms einen Interrupt; in der zugehörigen Interrupt-Routine wird dann die Systemzeit (Speicherzellen in Page Ø) inkrementiert. Für die Monitorzeitmessung kommt diese Möglichkeit nicht in Frage, da die Auflösung zu grob ist. Innerhalb der Realzeituhr ist jedoch ein 16 Bit Register vorhanden, das mit einer Frequenz von maximal 100 KHz hochgezählt wird (FRC Free Running Counter). Diese Auflösung genügt für die Zeitmessungen. Für die Zeitrechnung werden grundsätzlich Doppelworte verwendet. Damit ergibt sich in Abhängigkeit von der Frequenz eine Gesamtüberwachungszeit von

$$G = (2^{30}-1) \; / \; \text{Frequenz des FRC [sec]}$$

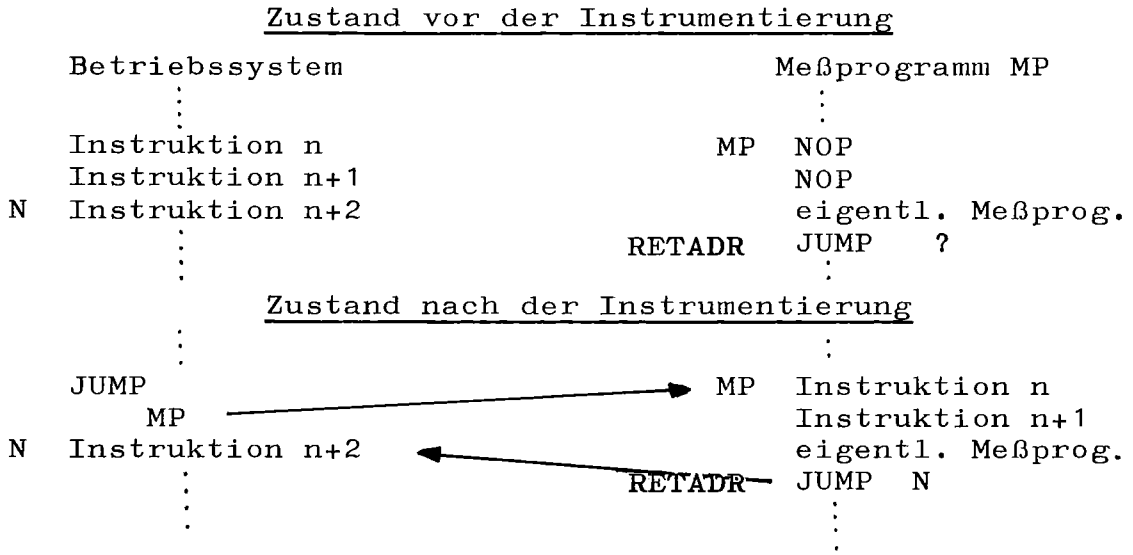
Um einen Überlauf des FRC nach 65535 Impulsen zu vermeiden, wird bei jedem Clock-Interrupt der Wert des FRC auf die Monitorzeit TIME addiert und anschließend gelöscht. Der FRC muß also nur die Impulse zwischen zwei Interrupts der Realzeituhr zählen, wofür seine Kapazität bei weitem ausreicht.
TIME enthält also die Zeit in Inkrementen der Realzeituhr bezüglich der Frequenz des FRC. Zur Ermittlung des momentanen Zeitpunkts, wird in der Meßroutine der Wert des FRC eingelesen und der Inhalt von TIME dazuaddiert; damit kann der exakte Zeitpunkt bis zu einer Genauigkeit von 10 µs bestimmt werden.



4.4 Generierung der Meßpunkte im Betriebssystem

Der Monitor führt seine Messungen ereignisgesteuert aus, d. h. bei Eintreffen eines bestimmten Ereignisses (Ende des EA-Vorgangs, Interrupt von Realzeituhr, Starten einer Task vom Dispatcher usw) muß ein spezifisches Meßprogramm aufgerufen werden. Da das Betriebssystem auch ohne den MONITOR benutzt werden soll, dürfen die Aufrufe von Meßprogrammen nicht fest im Betriebssystem installiert sein. Die Meßpunkte werden erst beim Start des MONITORS an die vorgesehenen Stellen eingesetzt.
Die Instrumentierung des Betriebssystems geschieht auf die folgende Art /8/:

An den Stellen, an denen eine Messung durchgeführt werden soll, wird der dort vorhandene Befehl gerettet und an dieser Adresse ein Sprung an das Meßprogramm eingetragen. Der gerettete Befehl wird an den Anfang oder das Ende der Meßroutine gestellt. Ist die Meßroutine abgearbeitet, so wird an den Befehl nach der Aufrufstelle zurückgesprungen.



JUMP = unbedingter Sprung an angegebene Adresse

NOP = Leerbefehl

N = Rücksprungadresse

RETADR siehe 4.7.5

Bild 4.3 Dynamische Instrumentierung des Betriebssystems

Dieses Konzept ist flexibel und kann in vielfältiger Weise den jeweiligen Gegebenheiten angepaßt werden.

Der Aufruf des Meßprogramms muß nicht durch einen unbedingten Sprung erfolgen, sondern kann auch abhängig vom Wert bestimmter Register sein. Die überschriebenen Befehle können am Anfang oder am Ende der Meßroutine ausgeführt werden. Jeder Meßpunkt wird durch einen Meßpunkt-Beschreibungs-Block definiert, in dem alle zur Generierung notwendigen Daten für diesen Meßpunkt enthalten sind (s. 4.7.5)

4.5 Generierung der Meßpunkte im Programm

Auf Programmebene beschränken sich die Messungen auf das Ermitteln der Zeitdauer und der Anzahl der Aufrufe von Meßbereichen. Diese Meßbereiche werden durch eine BEGIN-END-Klammerung festgelegt und mit einer Meßbereichsnummer versehen.

```
MBEGIN(5)
   :
   Programm-Statements
   :
MEND(5)
```

Eine Schachtelung bzw. eine Überlappung der Meßbereiche ist möglich.

Schachtelung

```
MBEGIN(1)
   :
MBEGIN(2)
   :
MEND(2)
   :
MEND(1)
   :
```

Überlappung

```
MBEGIN(1)
   :
MBEGIN(2)
   :
MEND(1)
   :
MEND(2)
   :
```

Mehrere Meßbereiche können mit der gleichen Meßbereichsnummer versehen sein. Die Zeitdauer für diese Meßbereiche wird dann gemeinsam akkumuliert.

Der Aufruf dieser BEGIN-END-Klammerung erfolgt über Unterprogramme, die als Parameter die Meßbereichsnummer übergeben.

```
          CALL   MBEGIN (Meßbereichnummer)
bzw.      CALL   MEND   (Meßbereichnummer)
```

Zusätzlich muß noch ein Datenfeld zur Aufnahme aller Meßbereichblöcke (s. 4.7.3) definiert werden. Die Länge dieses Feldes ist

(höchste Meßbereichsnummer + 1) * 7

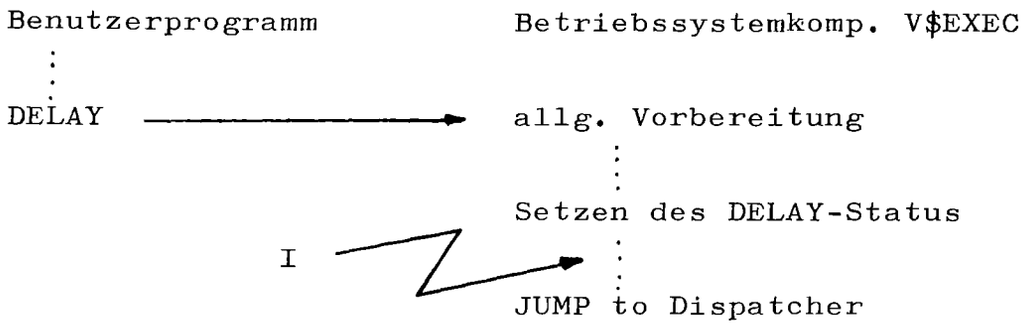
Die Anfangsadresse muß durch das Unterprogramm

```
CALL   MDEF (Feld)
```

vor Aufruf des ersten MBEGIN den Meßroutinen bekannt gemacht werden.

4.6 Abbildung des Taskzustandes

Neben der Messung von Rechenzeiten werden auch die Zeiten ermittelt, die eine Task in den unterschiedlichen Taskzuständen verbringt (s.2.5). Dazu muß das Taskzustandsmodell auf den MONITOR übertragen werden. In VORTEX ist es möglich, daß sich eine Task gleichzeitig in 2 Zuständen befindet. Dieser Tatbestand soll am Beispiel eines DELAY-Aufrufs verdeutlicht werden.



Die Befehle zwischen Setzen des DELAY-Status und dem Übergang an den Dispatcher sind nicht unterbrechungsfrei. Wird die Task nun hier durch den Interrupt I unterbrochen, so ergibt sich folgende Situation:

- Im Statuswort ist vermerkt, daß die Task auf den Ablauf einer Zeitspanne wartet. Sie wird also von der CLOCK-Routine bei der Herabsetzung der Verzögerungszeit berücksichtigt. Die Task befindet sich eindeutig im Zustand verzögert.
- Unter der Regie dieser Task müssen noch einige Befehle ausgeführt werden. Durch den Interrupt ist es möglich, daß zunächst eine Task mit höherer Prioritätbearbeitet wird. Die unterbrochene Task befindet sich dann aber im Zustand 'bereit', da sie weiterhin an der Vergabe des Prozessors beteiligt ist.

Einige solcher Situationen können vermieden werden, in dem die Befehlsfolgen (meist nur wenige Operationen) vor Unterbrechungen geschützt werden (bei Tasksteuerbefehlen DELAY usw). Dabei wird das Betriebssystem in seiner Funktion nicht beeinträchtigt. An anderen Stellen (EA-Komponente des Betriebssystems) kann dies nicht verhindert werden.

Um nun trotzdem eindeutige Zeitmessungen durchzuführen, muß die Beschreibung einer Task im Monitor durch 2 Zustände erfolgen:

Status 1 : Gibt die Programmteile an, in denen sich die Task gerade mit der Ausführung von Befehlen aufhält (Task ist im Zustand rechnend)

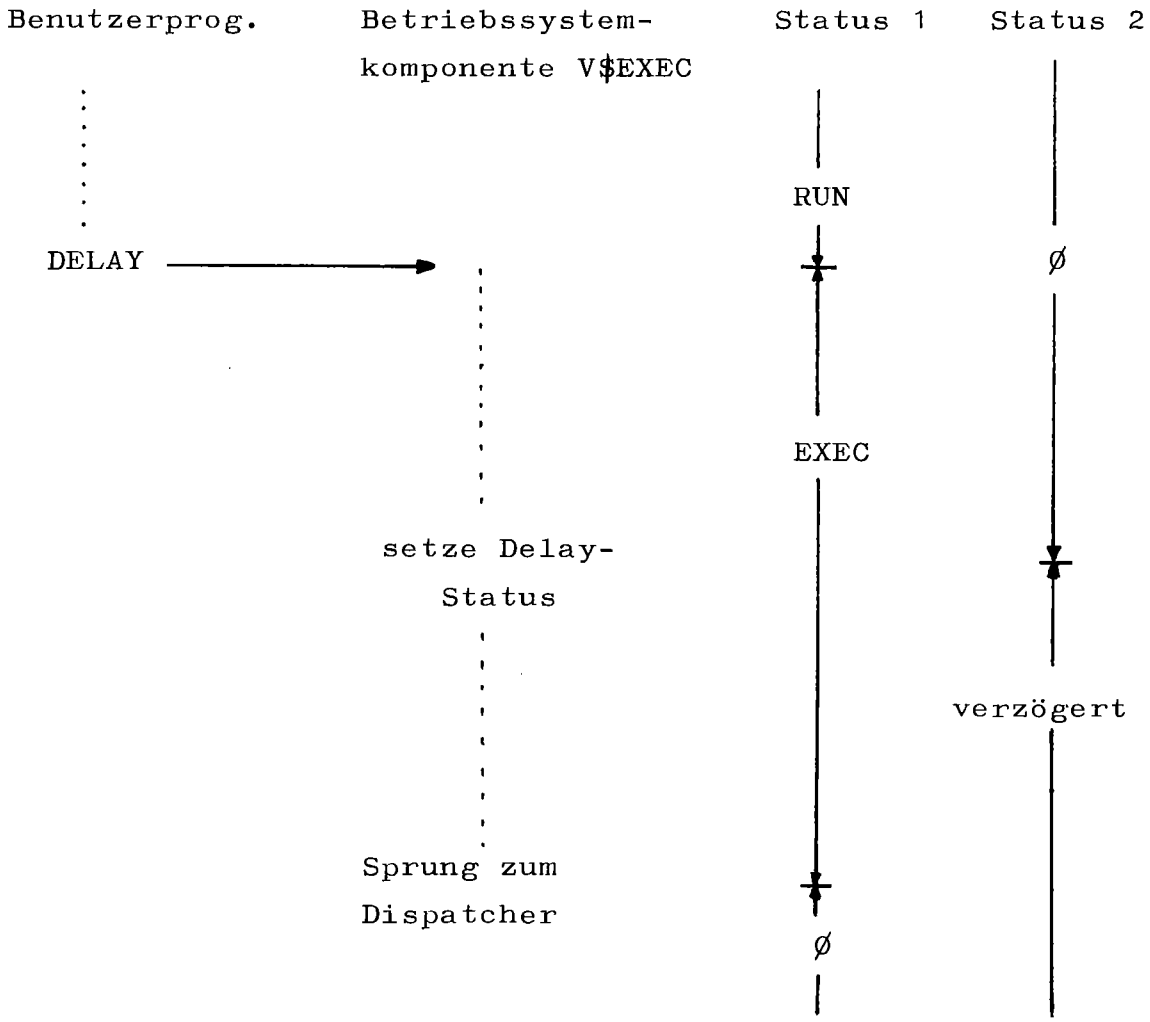
= RUN: Task führt Befehle des eigenen Programms aus

= EXEC:Task ist in der Bearbeitung von RTE-Service-Routinen

= IOC: Task ist in EA-Auftragsverwaltung

Status 2: Beschreibt alle Taskzustände, in denen die Task nicht über den Prozessor verfügt (Task nicht rechnend)

Am Beispiel des DELAY-Aufrufs ergibt sich folgendes Bild



Ø bei Status 2: Task ist rechnend

Ø bei Status 1: Task ist nicht rechnend

Bild 4.4 Belegung der 2 Taskzustandsanzeigen des Monitors am Beispiel des DELAY - Aufrufs

Wird eine Task unterbrochen, so wird vom MONITOR der Status nicht verändert. Da eine Task nur dann unterbrochen werden kann, wenn sie über den Prozessor verfügt und rechnet (Status 1 $\neq \emptyset$), kann bei Wiederaufnehmen der Bearbeitung eindeutig der alte Zustand rekonstruiert werden.

Status 1 $\neq \emptyset$, Status 2 $= \emptyset$

Task wurde unterbrochen. Die Zeitdauer für die Unterbrechung wird auf den Taskzustand 'bereit' addiert.

Status 1 $= \emptyset$, Status 2 $\neq \emptyset$

Normale Fortsetzung nach einer Suspendierung, Verzögerung etc.

Status 1 $\neq \emptyset$, Status 2 $\neq \emptyset$

Die Task wurde unterbrochen, als sie sich gleichzeitig in 2 Zuständen befand. Die Zeitdauer für die Unterbrechung wird schon bei der Zeit für den Zustand mitgezählt, der durch Status 2 angezeigt wird.

4.7 Datenstrukturen, Meßwerte

4.7.1 Der Monitor-Task-Block MTB (Messungen auf Taskebene)

Jede Anwendertask, die während der Monitorüberwachung gestartet wird, erhält einen eigenen Monitor-Task-Block zugewiesen. Dieser MTB enthält alle Meßdaten, die für diese Task gesammelt werden.

Die Zuordnung eines MTB zu einer Task wird durch den MAP-KEY beschrieben. Deshalb kann eine Task erst dann über einen MTB verfügen, wenn sie einen MAP-KEY besitzt. Die Abbildung des MAP-KEY auf den Monitor-Task-Block erfolgt über die Tabelle MTBTAB.

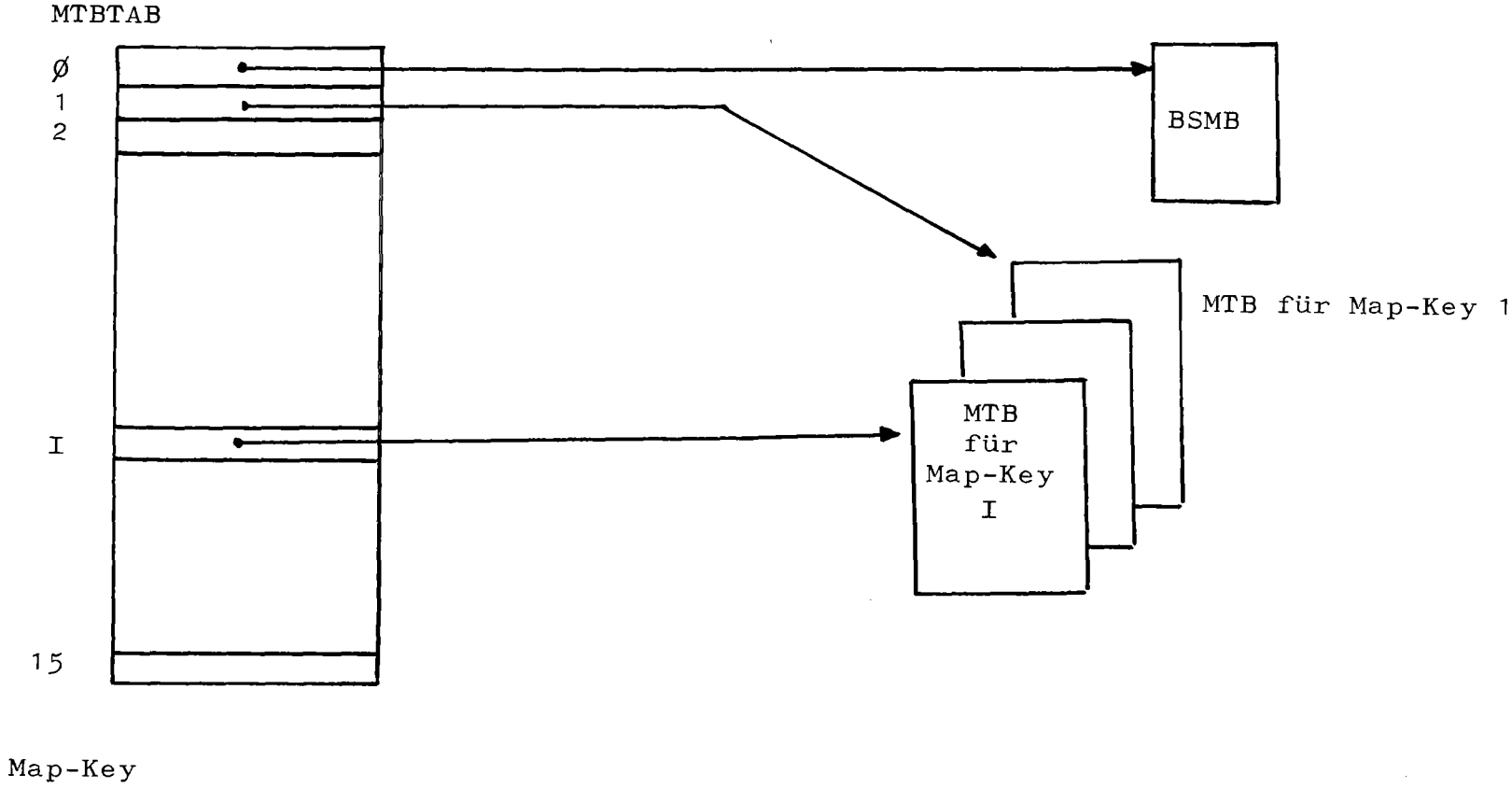


Bild 4.5 Identifizierung der Monitortaskblöcke über MAP-Key mit Hilfe von MTBTAB

Alle nicht belegten MTB sind in einer Freiliste über Wort Ø verkettet.

Die Meßdaten im MTB kommen folgendermaßen zustande:

Zeitdauer Task rechnend: Ausführungszeit für die in der Programmiersprache geschriebenen Befehle

Zeitdauer Task unterbrochen: Zeitdauer für Interrupt-Routinen, die während der Rechenzeit der Task eingeschoben wurden.

IOC-Zeit: Rechenzeit für die EA-Verwaltung.
EXEC-Zeit: Rechenzeit für RTE-Service-Routinen } von Task aufgerufen

Reentrant-Programme: Zeitdauer für die Benutzung von Reentrant Programmen (diese Zeit ist nicht im EXEC enthalten)

Gesamtreaktionszeit auf Interrupts:
Als Reaktionszeit gilt die Zeitspanne zwischen Eintragen des Interrupt-Ereignis-Worts und Aktivieren der Task. Diese Zeit wird nur ermittelt, falls die Task auf einen Interrupt wartet und geladen ist.

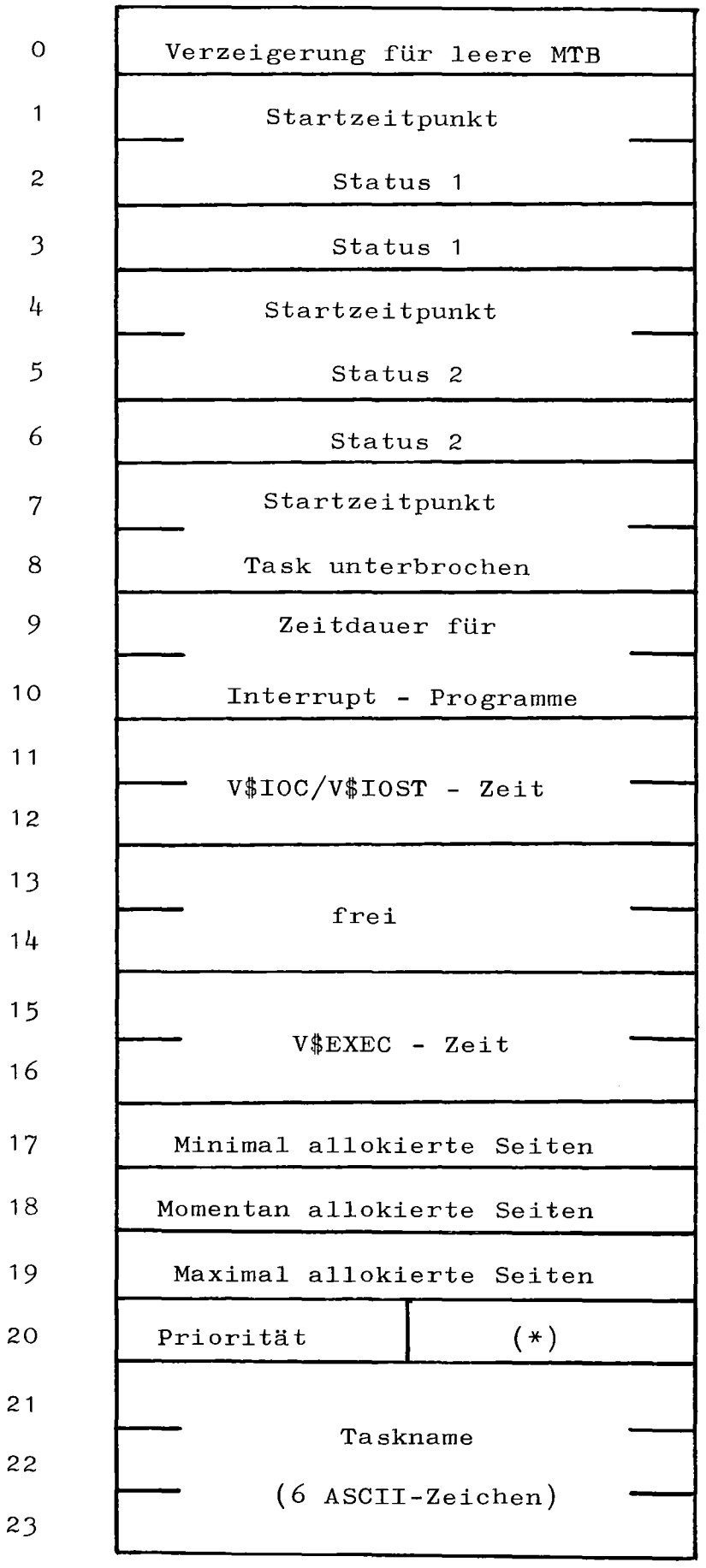
EA-Zeiten:

Nicht spezifizierte Geräte:
Gemeinsame EA-Zeiten für alle Geräte bzw. alle EA-Befehle, die nicht gesondert gemessen werden.

Parallele EA:
Zeitdauer für EA, auf deren Ende nicht gewartet wird.

Nicht parallele EA: (ab MTB-Zellen 54)
Zeiten für die einzelnen EA-Geräte.
Die EA-Zeit läuft ab dem Zeitpunkt, an dem der Treiber aktiviert wird, bis zum Löschen des Auftrags in der Auftragsliste.

Monitor - Task - Block MTB



(*) Anzahl der allokierten Seiten beim Laden

Bild 4.6 Aufbau eines Monitortaskblocks

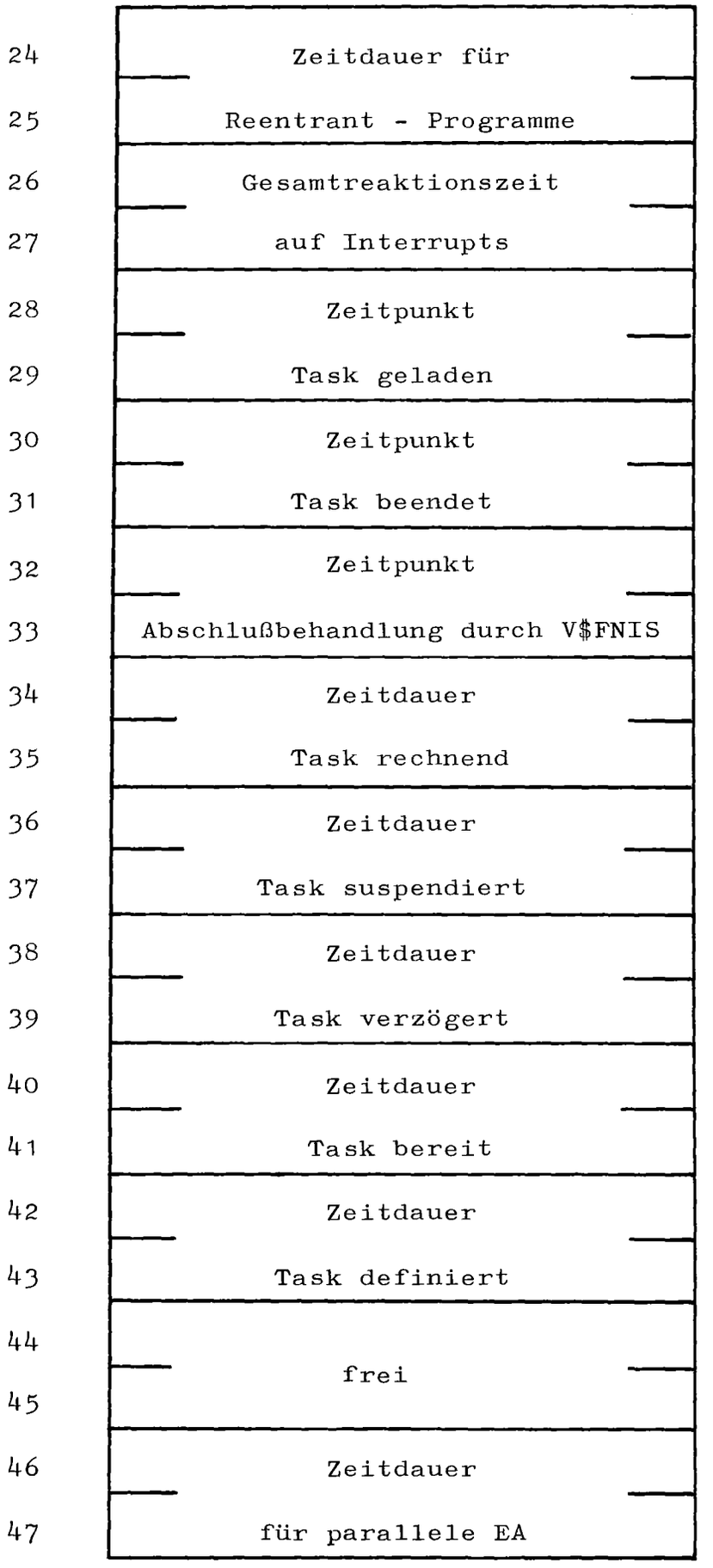
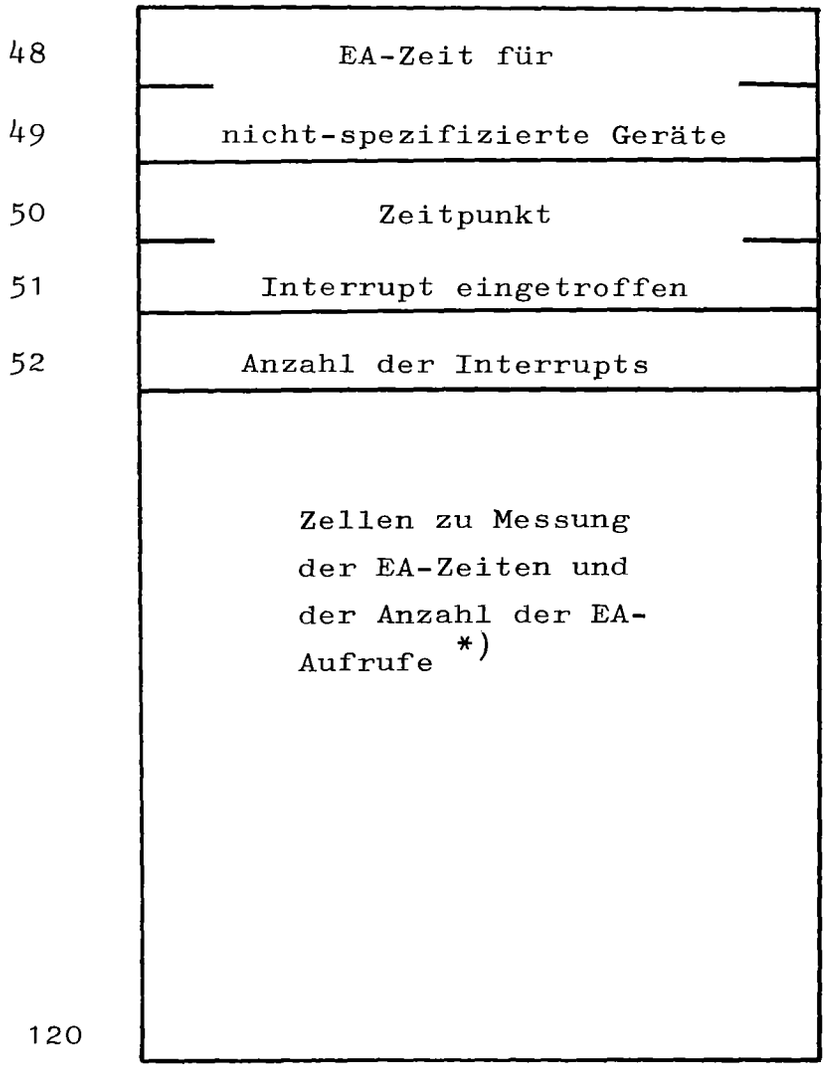


Bild 4.7 Aufbau eines Monitortaskblocks (Fortsetzung)



*) siehe 4.7.4

Bild 4.8 Aufbau eines Monitortaskblockes (Fortsetzung)

Diese Zeit ist nicht im Taskzustand 'suspendiert' enthalten, obwohl die Task bis zum Ende der EA suspendiert ist.

4.7.2 Der Betriebssystem-Monitor-Block BSMB (Messungen auf Betriebssystemebene)

Der Betriebssystem-Monitor-Block nimmt alle Meßdaten auf, die über die Ausführung des Betriebssystems ermittelt werden. Er ist nur einmal vorhanden.
Die Zeitdauer für die vom Betriebssystem initiierte Ein-Ausgabe wird in den für jedes Gerät angelegten Monitor-Controller Tabellen (s. 4.7.4) abgelegt.

Zeitdauer für Betriebssystem-Tasks.
Rechenzeiten für alle Map-Ø-Tasks (Treiber, Ladeprogramme, Fehlerausgabeprogramme, Operateurkommunikation, Interrupt-Routinen)

Dispatcher/Clock:
Rechenzeit für den Dispatcher und die Clock-Interrupt-Routine

Prozessor inaktiv: Zeitdauer, in der keine Programme bearbeitet werden

Kann der Dispatcher keine Task aktivieren, so durchläuft er so lange die Kette der Taskidentifikationsblöcke, bis er einer Task den Prozessor zuteilen kann. In dieser Zeit ist der Rechner eigentlich inaktiv in dem Sinne, daß keine Arbeit in unmittelbarem Zusammenhang mit den Aktivitäten einer Task erfolgt.

Deshalb gilt die Definition:
Der Prozessor geht in den Zustand inaktiv über, wenn nach einmaligem Durchlaufen der TIDB-Kette keine Task aktiviert werden konnte.

Nach einem Interrupt wird dieser Zustand wieder verlassen.

Betriebssystem - Monitor - Block BSMB

0	Anzahl d. Power-failure-Interrupts
1	Startzeitpunkt
2	für Dispatcher und Clock
3	Endezeitpunkt
4	Monitorbetrieb (Systemzeit)
5	Zeitdauer
6	Monitor aktiv (Monitorzeit)
7	Startzeitpunkt
8	Task unterbrochen
9	Zeitdauer
10	für Betriebssystem-Tasks
11	Zeitdauer
12	für Dispatcher und Clock
13	Startzeitpunkt
14	für Betriebssystem-Tasks
15	Startzeitpunkt
16	Monitorbetrieb (Systemzeit)
17	Minimal allokierte Seiten
18	Momentan allokierte Seiten
19	Maximal allokierte Seiten
20	frei
21	
22	'VORTEX'
23	
24	Zeitdauer
25	CPU inaktiv

Bild 4.9 Aufbau des Betriebssystem-Monitor-Blocks

4.7.3 Der Meßbereichsdatenblock
(Messungen auf Programmebene)

Bei Programm-Messungen wird für jeden Meßbereich ein Datenblock mit folgenden Informationen angelegt:

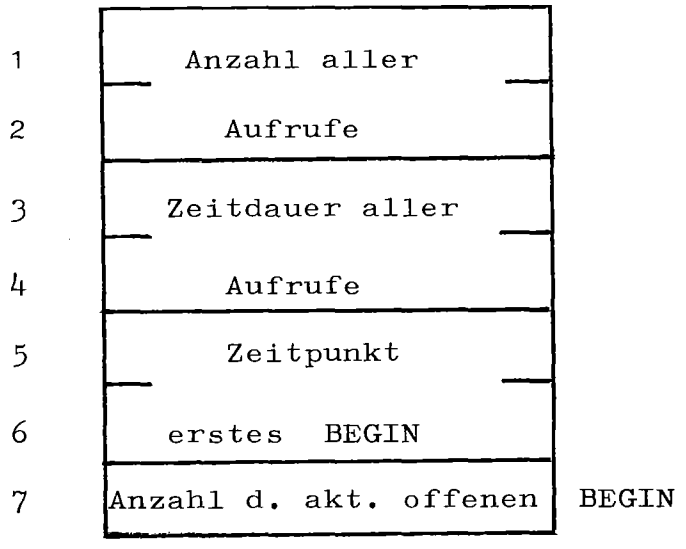


Bild 4.10 Aufbau eines Meßbereichsdatenblocks

Beim ersten BEGIN wird der Zeitpunkt in den Worten 5 und 6 des zugehörigen MTB eingetragen und die Anzahl der aktuell offenen BEGIN auf 1 gesetzt. Jedes weitere BEGIN erhöht diesen Zähler, jedes END vermindert ihn um 1. Ist die Anzahl der BEGIN = Ø, so wird die Zeitdauer zwischen dem ersten BEGIN und diesem Punkt gemessen und auf die Worte 3 und 4 addiert. Die Anzahl aller Aufrufe wird inkrementiert.

4.7.4 Monitor Controller Tabelle MCTBL

Für jedes an den Rechner angeschlossene EA-Gerät muß zur Beschreibung der EA-Überwachung ähnlich der Controller-Tabelle (s.2.3) eine Monitor-Controller-Tabelle angelegt werden. Damit kann der Benutzer genau definieren, für welche EA-Geräte bzw. für welche EA-Befehle eines Peripheriegerätes die Dauer der Ein-Ausgabe ermittelt werden soll.

Monitor - Controller - Tabelle MCTBL

0	(*)	(**)	READ
1	(*)	(**)	WRITE
2	(*)	(**)	WEOF
3	(*)	(**)	REW
4	(*)	(**)	SREC
5	(*)	(**)	FUNC
6	(*)	(**)	OPEN
7	(*)	(**)	CLOSE
8	Startzeitpunkt des		
9	momentanen EA-Vorgangs		
10	EA-Zeit des Betriebssystems		
11	für dieses Gerät		
12			
13	Gerätename		
14	(8 ASCII-Zeichen)		
15			
16	Status		

(*) Zelle in MTB, in der die Zeitdauer für diesen EA-Befehl akkumuliert wird

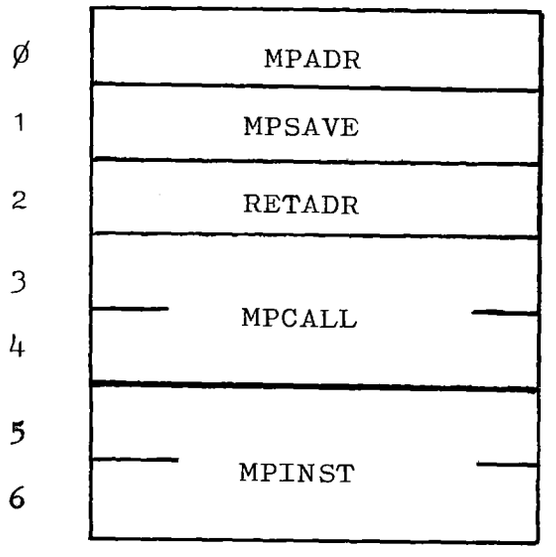
(**) MTB-Zelle, in der die Anzahl der Aufrufe für diesen EA-Befehl gezählt wird

siehe S.44

Bild 4.11 Aufbau einer Monitor-Controller-Tabelle

4.7.5 Meßpunkt-Beschreibungs-Block

Jeder Meßpunkt muß durch einen Meßpunkt-Beschreibungs-Block definiert werden. Dieser Datenblock enthält folgende Informationen:



MPADR : Adresse des Meßpunktes
MPSAVE: Adresse, an der die durch den Aufruf des Meßprogramms überschriebenen Befehle gerettet werden (NOP-Befehle in Meßroutine)*)
RETADR: Adresse, an der Rücksprungadresse abgelegt werden soll *)
MPCALL: Aufruf für zugehöriges Meßprogramm
MPINST: hier werden die durch den Aufruf des Meßprogramms überschriebenen Befehle gerettet

*) siehe 4.4

Bild 4.12 Aufbau eines Meßpunktbeschreibungsblocks

4.8 Generieren und Aktivieren des Monitors

Obwohl die Messungen im Betriebssystem durchgeführt werden, sind die Meßprogramme und die Monitor-Tabellen nicht ständig im Betriebssystem-Bereich vorhanden. Durch eine speziell entwickelte Schnittstelle ist es möglich, den Monitor nachträglich in das Betriebssystem einzufügen. Bei der Systemgenerierung muß nur der Speicherbereich für das Laufzeitmeßsystem reserviert werden.

Zur Generierung des Monitors sind die nachstehenden Schritte notwendig:
Der Monitor wird zunächst gemeinsam mit dem Ladeformat-Generierungsprogramm übersetzt und als Task gebunden ① . Nach dem Starten dieser Task erzeugt das Ladeformat-Programm die Ladeform des Monitors ② . Dabei können über die Operateurkonsole bestimmte Varianten des Monitors erzeugt werden:

1) alle Meßpunkte werden generiert
2) EA-Messungen werden nicht durchgeführt
3) EXEC-Messungen werden nicht durchgeführt
4) EA- und EXEC Messungen fallen weg
5) Minimal-Version

Je mehr Messungen wegfallen, umso weniger Informationen werden über die Ausführungszeit der einzelnen Tasks gesammelt. Dies geht soweit, daß in der Minimal-Version des Monitors nur noch Messungen auf Betriebssystemebene vorgenommen werden und die Unterscheidung in Benutzertasks und Betriebssystemtasks entfällt. Gleichzeitig nimmt auch die Belastung des Rechners durch den Monitor ab, was auch die Verfälschung der Meßergebnisse reduziert. Damit ist die Generierungsphase abgeschlossen.

Zum Aktivieren des Monitors wird zuerst die Ladeform mit Hilfe eines Ladeprogramms in den für das Laufzeitmeßsystem reservierten Speicherbereich geladen ③ .
Zum Instrumentieren des Betriebssystems ist nun ledig-

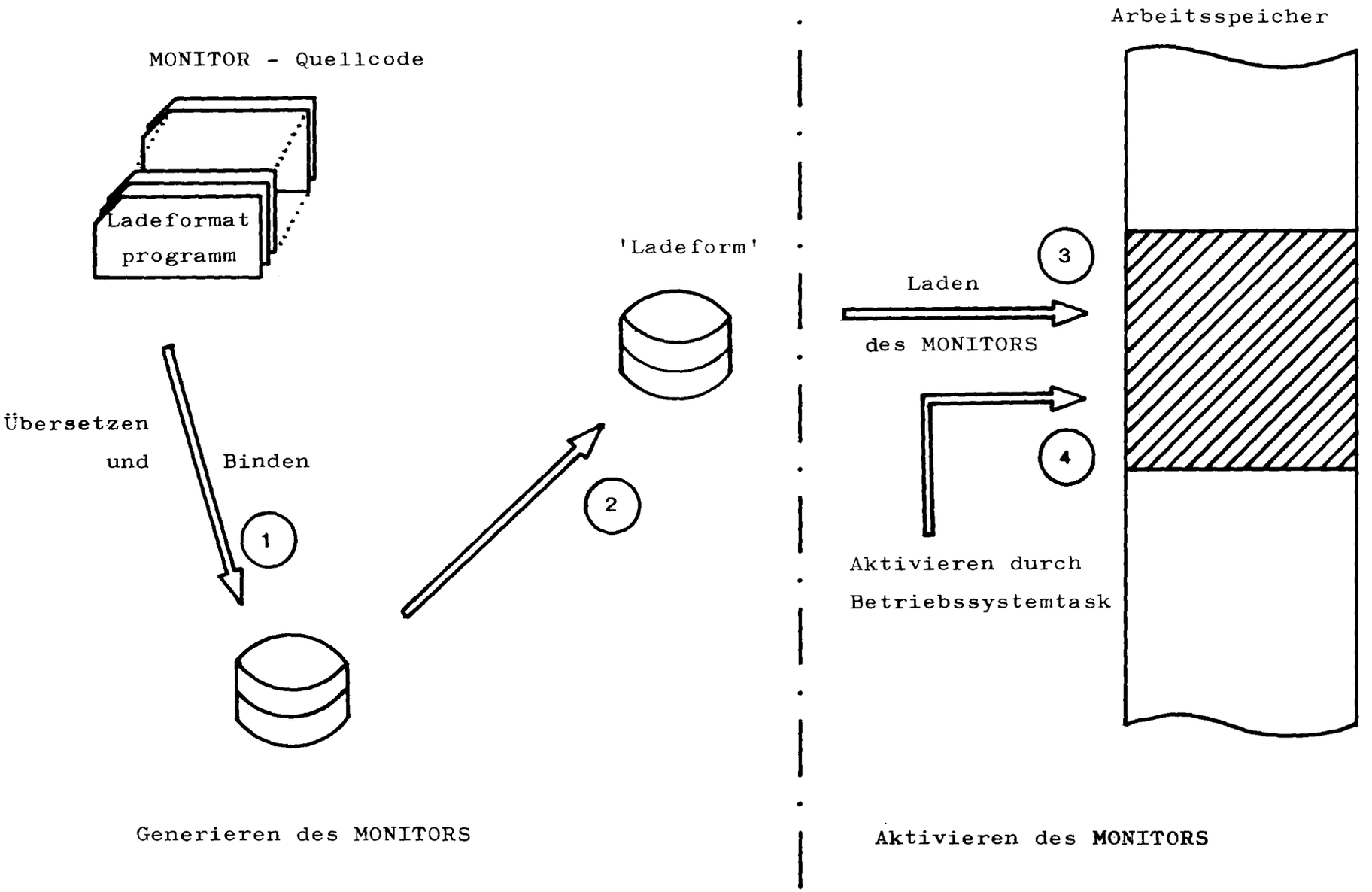


Bild 4.13 Generieren und Aktivieren des MONITORS

lich eine speziell entwickelte Systemtask zu starten, die die Meßpunkte in das Betriebssystem einfügt ④. Anschließend können die Messungen durchgeführt werden. Der Monitor weist jeder neu gestarteten Anwendertask einen Monitor-Task-Block zu. Bei Ende der Task wird der MTB mit den Meßdaten auf Platte ausgelagert. Durch erneuten Aufruf der oben erwähnten Systemtask wird der Monitor abgebrochen und die Meßpunkte aus dem Betriebssystem entfernt. Der Betriebssystemmonitor-block und die Monitor-Controller-Tabellen werden anschließend ebenfalls auf die Plattendatei ausgegeben. Von dort können die Meßwerte durch ein spezielles Auswerteprogramm auf Drucker ausgegeben werden. Ein Beispiel für eine solche Druckerausgabe ist auf den folgenden Seiten angegeben:

4.9 Kenndaten des Monitors

Der Monitor ist in Assembler geschrieben und benötigt einschließlich der 15 Monitor-Task-Blöcke à 120 Worte 5 K Speicher.

Die Belastung des Betriebssystems durch das Ausführen der Meßprogramme zerfällt in eine konstante Belastung durch den regelmäßigen Aufruf von instrumentierten Programmen (Clock-Interrupt-Routine, Dispatcher); sie ist < 1 % (rechnerisch durch Abzählen der Befehle ermittelt). Die Belastung durch die anderen Meßprogramme steigt linear mit der Anzahl der aufgerufenen Meßroutinen. Durchschnittlich werden für ein Meßprogramm 40 µs bei Anwendertasks und 15 µs bei Systemtasks benötigt (rechnerisch ermittelt).

```
          ****************
          *              *
TASK      *    ALARM1    *   PRIORITAET:  17
          *              *
          ****************


STARTZEITPUNKT                        0 H  0 MIN 26.501100 SEC
ENDEZEITPUNKT                         0 H  2 MIN 46.883500 SEC
ABSCHLUSS                             0 H  2 MIN 46.885600 SEC
ZEITDAUER TASK IM SPEICHER            0 H  2 MIN 20.384500 SEC


TASKGROESSE:
------------

BEIM LADEN                          24 SEITEN
DYNAMISCH    MINIMAL                24 SEITEN
             MAXIMAL                24 SEITEN


ZEITEN FUER TASKZUSTAENDE:
--------------------------

DEFINIERT                  0 H  0 MIN  00.0000* SEC
BEREIT                     0 H  0 MIN  2.005500 SEC
RECHNEND                   0 H  0 MIN  1.543000 SEC
SUSPENDIERT                0 H  1 MIN 56.926100 SEC
VERZOEGERT                 0 H  0 MIN  00.0000* SEC
UNTERBROCHEN               0 H  0 MIN  00.0000* SEC

TOTAL 1                    0 H  2 MIN  0.474600 SEC


ZEITEN FUER BETRIEBSSYSTEM-PROGRAMM-AUFRUFE:
--------------------------------------------

V$IOC                      0 H  0 MIN  1.247000 SEC
V$EXEC                     0 H  0 MIN  0.256900 SEC
REENTRANT PROGRAMME        0 H  0 MIN  00.0000* SEC

TOTAL 2                    0 H  0 MIN  1.503900 SEC


ZEITEN FUER EIN/AUSGABE:
------------------------

NICHT SPEZIF. GERAETE      0 H  0 MIN  00.0000* SEC
NICHT PARALELLE E/A        0 H  0 MIN 15.801900 SEC

TOTAL 3                    0 H  0 MIN 15.801900 SEC
PARALELLE E/A              0 H  0 MIN  00.0000* SEC



TOTAL 1 + TOTAL 2 + TOTAL 3 =
GESAMTZEIT                 0 H  2 MIN 17.780400 SEC

DURCHSCHNITTLICHE REAKTIONSZEIT BEI   155 INTERRUPTS:
                           0 H  0 MIN  0.000887 SEC
```

Bild 4.14 Ausgabeprotokoll für Taskmessungen

```
ZEITEN FUER DIE EIN/AUSGABE AUFGESCHLUESSELT FUER DIE
VERSCHIEDENEN E/A-GERAETE
(SCHLUESSEL SIEHE E/A-TABELLE)

     56:       0 H   0 MIN   00.0000* SEC
     66:       0 H   0 MIN   00.0000* SEC
     78:       0 H   0 MIN   00.0000* SEC
     72:       0 H   0 MIN   0.650600 SEC
     70:       0 H   0 MIN   00.0000* SEC
     60:       0 H   0 MIN   00.0000* SEC
     96:       0 H   0 MIN   1.370200 SEC
     62:       0 H   0 MIN   1.848500 SEC
     76:       0 H   0 MIN   9.797400 SEC
     54:       0 H   0 MIN   00.0000* SEC
     64:       0 H   0 MIN   1.375700 SEC
     74:       0 H   0 MIN   00.0000* SEC
     58:       0 H   0 MIN   00.0000* SEC
     68:       0 H   0 MIN   0.759500 SEC


     82:             0 AUFRUFE
     87:             0 AUFRUFE
     94:             0 AUFRUFE
     95:             0 AUFRUFE
     91:           731 AUFRUFE
     90:             0 AUFRUFE
     84:             0 AUFRUFE
     80:            36 AUFRUFE
     85:            54 AUFRUFE
     93:           180 AUFRUFE
     81:             0 AUFRUFE
     86:            27 AUFRUFE
     92:             0 AUFRUFE
     83:             0 AUFRUFE
     88:             3 AUFRUFE
```

Bild 4.15 Ausgabeprotokoll für Taskmessungen (Fortsetzung)

E/A-TABELLE:

	READ	WRITE	WEOF	REW	SREC	FUNC	OPEN	CLOSE
WCS	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1
VFD	56/82	66/87	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1
TEKTRONX	56/82	66/87	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1
MOPEK	78/94	78/95	0/-1	72/91	0/-1	0/-1	0/-1	0/-1
PAP.TAPE	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1
DRUCKER	0/-1	70/90	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1
KARTENL.	60/84	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1
PLATTE	96/80	62/85	0/-1	0/-1	0/-1	0/-1	76/93	76/93
MAG.TAPE	54/81	64/86	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1
CMA	0/-1	74/92	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1
TELETYPE	58/83	68/88	0/-1	0/-1	0/-1	0/-1	0/-1	0/-1

ERKLAERUNG:

Z/A BEDEUTET
Z: E/A-ZEIT IST AUFGEFUEHRT UNTER
SCHLUESSELNUMMER Z
A: ANZAHL DER AUFRUFE IST AUFGEFUEHRT
UNTER SCHLUESSELNUMMER A

BESONDERE SCHLUESSELNUMMERN:
0: E/A-ZEIT IST ENTHALTEN IN E/A-ZEIT FUER
NICHT SPEZIFIZIERTE E/A-GERAETE
-1: E/A WIRD NICHT UEBERWACHT

Bild 4.16 Erläuterung der Schlüssel für EA-Zeiten bei Taskmessungen (s. S. 54)
(Ausgabeprotokoll)

```
****************
*              *
*    VORTEX    *
*              *
****************
```

```
MESSUNGEN FUER DAS BETRIEBSSYSTEM:
----------------------------------

STARTZEITPUNKT DER MONITORUEBERWACHUNG     2 H 43 MIN  2.621094 SEC
ENDEZEITPUNKT DER MONITORUEBERWACHUNG      2 H 45 MIN 55.820313 SEC
ZEITDAUER MONITOR AKTIV                    0 H  2 MIN 53.191200 SEC

GROESSE DES BETRIEBSSYSTEMS:
----------------------------

NACH BOOT                 53 SEITEN
DYNAMISCH   MINIMAL       53 SEITEN
            MAXIMAL       53 SEITEN

ZEITMESSUNGEN:
--------------

BETRIEBSSYSTEM-TASKS       0 H  0 MIN  4.422900 SEC
DISPATCHER / CLOCK         0 H  0 MIN 10.323700 SEC
CPU INAKTIV                0 H  2 MIN 32.840000 SEC
BENUTZER-TASKS             0 H  0 MIN  5.604600 SEC
```

Bild 4.17 Ausgabeprotokoll für Betriebssystemmessungen

E/A-ZEITEN FUER DAS BETRIEBSSYSTEM:

WCS	0	H	0	MIN	00.0000* SEC
VFD	0	H	0	MIN	00.0000* SEC
TEKTRONX	0	H	0	MIN	00.0000* SEC
MOPEK	0	H	0	MIN	00.0000* SEC
PAP.TAPE	0	H	0	MIN	00.0000* SEC
DRUCKER	0	H	0	MIN	00.0000* SEC
KARTENL.	0	H	0	MIN	00.0000* SEC
PLATTE	0	H	0	MIN	2.715200 SEC
MAG.TAPE	0	H	0	MIN	00.0000* SEC
CMA	0	H	0	MIN	00.0000* SEC
TELETYPE	0	H	0	MIN	00.0000* SEC

Bild 4.18 Ausgabeprotokoll für Betriebssystemmessungen (Fortsetzung)

5. Vergleichsmessungen

Mit dem Monitor wurden Messungen an einem Prozeßüberwachungssystem vorgenommen, das in Assembler, FORTRAN, IFTRAN und PASCAL geschrieben wurde /1/.

5.1 Testumgebung

Das Prozeßüberwachungssystem überwacht und steuert ein Differenzdichtemeßgerät zur Bestimmung von Uran-Konzentrationen. Der Meßvorgang erfolgt zyklisch und gliedert sich in 3 Teile

- Ansaugen der Meßlösungen
- Meßwert bestimmen
- Ablassen der Meßlösungen

Das Programmsystem zur Durchführung dieser Aufgaben besteht bei ASSEMBLER, FORTRAN und IFTRAN aus 2 Tasks:

ALARM 3 Alarmsystem, reagiert auf Interrupts der induktiven Meßgeber (Füllstandsanzeige) des Differenzdichtemeßgerätes

ALARM 1 Ablaufsteuerung
übernimmt die zeitliche Überwachung der 3 Phasen durch Öffnen und Schliessen von Ventilen, liest die Meßwerte ein und gibt sie aufbereitet auf Magnetband aus. Ist die Zeit zwischen den Meßvorgängen $\geq$ 50 sec, so werden die Meßwerte zusätzlich auf Drucker und der Zustand des Meßgerätes auf einen halbgraphischen Bildschirm (VFD Video Farb Display) ausgegeben.

Da in PASCAL kein Multitasking möglich ist, werden die Aufgaben von ALARM 3 innerhalb der Ablaufsteuerung ALARM 1 erfüllt, sodaß hier nur eine Task existiert.

Der reale Prozeß stand im Prozeßrechnerversuchsfeld für Testzwecke nicht zur Verfügung. Er wurde deshalb durch ein in Assembler geschriebenes Simulationssystem (ALARM2) nachgebildet, das unter allen vorhandenen Tasks mit der höchsten Priorität (30) läuft. Die Kommunikation zwischen Simulationssystem und Prozeßüberwachung erfolgt dabei konsequent über das Prozeßinterface.

Für jede Programmiersprache wurden 9 Messungen durchgeführt:

40 Zyklen à 2 sec Zykluszeit
80 Zyklen à 2 sec Zykluszeit
40 Zyklen à 3 sec Zykluszeit
80 Zyklen à 3 sec Zykluszeit
40 Zyklen à 5 sec Zykluszeit
80 Zyklen à 5 sec Zykluszeit
10 Zyklen à 50 sec Zykluszeit
20 Zyklen à 50 sec Zykluszeit
30 Zyklen à 50 sec Zykluszeit

die im weiteren mit 40/2, 80/2, 40/3 .....20/50,30/50 bezeichnet werden.
Als Zykluszeit ist dabei die maximale Zeitspanne zwischen 2 aufeinanderfolgenden Dichtemessungen zu verstehen; in der Praxis werden diese Zeiten jedoch unterboten. Dies wirkt sich aber nicht auf die Rechenzeit der Programme aus, da diese Zykluszeit durch die Totzeiten (Suspendierung, Verzögerung) bestimmt wird /1,S.34/.

Durch Abschalten des Zufallszahlengenerators im Simulationssystem waren für alle Programme gleiche Voraussetzungen gegeben.

Es wurden keine EA-Messungen (Anzahl der EA-Aufträge pro Task, Dauer der jeweiligen EA-Vorgänge) vorgenommen, um die Verfälschung der Meßergebnisse durch eine verminderte Monitorbelastung geringer zu halten.

Die Genauigkeit der Zeitmessung betrug 10^{-4} sec
($\hat{=}$ Frequenz des FRC = 10 kHz).

Die in den Tabellen im Anhang angegeben Meßzeiten sind auf 2 Stellen nach dem Komma gerundet. Teilweise wurde mit diesen gerundeten Werten weitergerechnet, sodaß kleine Abweichungen in der letzten Stelle auftreten können.

Für die Vergleichsgrundlage sind weniger die absoluten Rechenzeiten interessant, da diese problemabhängig sind. Wichtiger ist das Verhältnis der Rechenzeiten zwischen den einzelnen Programmiersprachen. Die absoluten Zeiten sind nur zur Dokumentation im Anhang mit aufgeführt.

Für BASIC konnten keine Messungen gemacht werden, da hier das Prozeßüberwachungssystem aus schon erwähnten Gründen (nur Background-Task, keine Reaktionsmöglichkeit auf Interrupts) nicht implementiert wurde.

5.2 Programmrechenzeiten

Für die Messung der Ausführungszeiten von Programmteilen des Überwachungssystems wurde das Unterprogramm MDRECH (s. Anhang) ausgewählt, das die Umrechnung der eingelesenen Meßwerte vornimmt. Da der Aufwand für eine BEGIN-END Messung im Verhältnis zur Rechenzeit des Prozeßüberwachungssystems in diesem Fall zu groß gewesen wäre, fanden die Messungen für das Unterprogramm getrennt statt. MDRECH wurde über eine Schleife 10000 und 30000 mal aufgerufen. Nach Abzug der Rechenzeit für die Schleife ergaben sich sehr genaue Meßwerte.

ASSEMBLER	FORTRAN	IFTRAN	PASCAL
1	2.55	2.55	5.58

Verhältnis der Rechenzeiten von MDRECH zu Assembler

Dieses Ergebnis war zu erwarten und es bestätigt sich auch in späteren Messungen (5.3). Der große Abstand von PASCAL beruht auf der Tatsache, daß der Compiler keinen Objektcode auf Maschinenniveau erzeugt, sondern einen Zwischencode generiert, der anschließend vom PASCAL-Interpreter ausgeführt wird.

Für eine Beurteilung ist diese Messung jedoch unzureichend da sie nur einen Teil der Belastung des Rechners ausmacht. Durch die intensive EA-Tätigkeit und die Aufrufe zur Tasksteuerung entsteht ein nicht unerheblicher Aufwand, der ebenfalls zu berücksichtigen ist.

5.3 Messungen auf Taskebene

Die in den folgenden Abschnitten durchgeführten Messungen wurden mit dem Monitor während der verschiedenen Meßzyklen gemacht. Dabei gilt als "Rechenzeit" diejenige Zeit, die zum Ausführen der in der Programmiersprache geschriebenen Befehle nötig war. Bei den höheren Programmiersprachen ist hierin auch der Aufwand für das Laufzeitsystem enthalten.

	ASSEMBLER	FORTRAN	IFTRAN	PASCAL
ohne VFD	1	2,35	2.35	5.80
mit VFD	1	1.93	1.96	6.28

Verhältnis der Rechenzeiten bzgl. Assembler

Für die Verhältnisse der Rechenzeiten ergeben sich etwa die gleichen Werte wie aus 5.2. Die Zeit für IOC und EXEC Aufrufe ist im großen und ganzen gleich, da jedes Prozeßüberwachungssystem die gleichen Anforderungen erfüllen soll. Die Programme für RTE-Routinen und EA-Verwaltung liegen im Betriebssystem und sind somit für alle Programmiersprachen identisch.

Aufschlußreich ist aber das Verhältnis von Rechenzeit zu EXEC und IOC-Zeit.

	ASSEMBLER	FORTRAN	IFTRAN	PASCAL
ohne VFD	.35	.83	.84	2.01
mit VFD	.32	.75	.77	2.60

Verhältnis Rechenzeit / (IOC + EXEC)

Aus dieser Tabelle geht hervor, daß der Aufwand für Betriebssystemdienste mit Ausnahme von PASCAL bis zum Faktor 3 größer ist als die eigentliche Rechenzeit. Dies bedeutet, daß der größte Teil der Rechnerbelastung durch Programmteile des Betriebssystems erfolgt, auf deren Programmierung der Anwender keinen Einfluß hat.

Nimmt man zur Rechenzeit diese Zeit für Betriebssystem-routinen mit hinzu und definiert als

Gesamtrechenzeit einer Task: = Rechenzeit + EXEC + IOC,

so ergeben sich für die untersuchten Sprachen die folgenden Werte:

	ASSEMBLER	FORTRAN	IFTRAN	PASCAL
mit VFD	1	1.30	1.30	2.19
ohne VFD	1	1.10	1.10	2.18

Verhältnis der Gesamtrechenzeit zu Assembler

Der Unterschied (besonders bei Ausgabe mit VFD) ist zwischen FORTRAN/IFTRAN und ASSEMBLER nur noch gering. Selbst bei PASCAL ist der Abstand zu Assembler stark zurückgegangen, aber trotzdem noch um den Faktor 2 größer.

5.4 Messungen auf Betriebssystem-Ebene

Die Messungen auf Betriebssystem-Ebene erfolgen ohne Unterscheidung der einzelnen Tasks. Die Meßwerte setzen sich folgendermaßen zusammen:

Betriebssystem-Tasks:	Zeitdauer für Lader (V$SAL), Interrupt-Prozessoren, Treiberprogramme
Dispatcher-Overhead:	Zeitdauer Dispatcher/Clock - Zeitdauer für Grundbelastung Dispatcher/Clock
Dispatcher/Clock Grundbelastung:	Zeitdauer Monitor aktiv *.04501 (≙ 4,5 % der Zeit die der Monitor aktiv war. Dieser Wert wurde experimentell bei Leerlauf des Rechners ermittelt).
Benutzertasks:	Gesamtrechenzeit für
	ALARM 1
	ALARM 3 (nicht bei PASCAL)
	ALARM 2 (Simulationssystem)
	PAREIN, INPAS, INABST } Initialisierungstasks für Prozeßüberwachung
	INITSS Initialisierungstask für Simulationssyst.

Bei der Gesamtrechenzeit der Benutzertasks bestätigt sich wieder das Ergebnis aus 5.3, obwohl hier auch andere Tasks (z.B. Simulationssystem in Assembler) mitzählen

	ASSEMBLER	FORTRAN	IFTRAN	PASCAL
ohne VFD	1	1.40	1.40	1.95
mit VFD	1	1.10	1.10	1.90

Verhältnis Benutzer-Task-Gesamtrechenzeiten zu Assembler

Für Betriebssystem-Tasks und Dispatcher-Overhead ergeben sich die folgenden Verhältnisse:

	ASSEMBLER	FORTRAN	IFTRAN	PASCAL
ohne VFD	1	1.20	1.25	1.25
mit VFD	1	.93	.98	.83

Verhältnis Betriebssystemtasks zu Assembler

	ASSEMBLER	FORTRAN	IFTRAN	PASCAL
ohne VFD	1	1.10	1.10	.96
mit VFD	1	.91	.90	.45

Verhältnis Dispatcher-Overhead zu Assembler

Aus den beiden letzten Tabellen geht hervor, daß alle Sprachen etwa den gleichen Aufwand an Betriebssystemaktivitäten verursachen. Er wird hauptsächlich durch die starke Ein-Ausgabe-Tätigkeit hervorgerufen.

Untersucht man nun das Verhältnis der Rechenzeiten der Benutzertasks zu denen für Betriebssystem-Tasks und Dispatcher-Overhead, so stellt man fest, daß hier eine sehr große Belastung vom Betriebssystem verursacht wird.

	ASSEMBLER	FORTRAN	IFTRAN	Pascal
ohne VFD	1.70	1.90	1.90	2.80
mit VFD	.38	.46	.48	.88

Verhältnis Benutzertasks/Betriebssystemtasks

	ASSEMBLER	FORTRAN	IFTRAN	PASCAL
ohne VFD	2.10	2.68	2.68	4.20
mit VFD	1.34	1.62	1.63	6.00

Verhältnis Benutzertasks/Dispatcher-Overhead

Wenn der Zustand des überwachten Prozesses graphisch auf einem Bildschirm angezeigt wird ist die Belastung des Rechners durch Programme des Betriebssystems beträchtlich. Sie ist bei allen Sprachen größer als die Belastung durch Benutzerprogramme.

Selbst der Dispatcher-Overhead, also die zusätzliche Zeit für das Umschalten von Tasks, ist erstaunlich groß gegenüber den Benutzerprogrammrechenzeiten. Dies ist auf das ungünstige Ablegen der Taskbeschreibungsblöcke in einer linearen, nach Prioritäten geordneten Liste zurückzuführen.

5.5 Reaktionszeiten auf Interrupts

Während der Messungen wurde auch die durchschnittliche Reaktionszeit auf Interrupts ermittelt. Dies ist die Zeit zwischen dem Auftreten des Interrupts (Start des Interrupt-Prozessors) und dem Aktivieren der auf diesen Interrupt wartenden Task.

Die Reaktionszeit für ALARM 2 (Simulationssystem) und ALARM 3 ist dabei bei allen Messungen konstant. Das Simulationssystem arbeitet mit der höchsten Priorität (30). Seine Reaktionszeit kann daher als "Minimum" angesehen werden, da diese Task sofort nach Auftreten des Interrupts gestartet und die Reaktionszeit nur durch die Verwaltungszeit des Betriebssystems bestimmt wird.

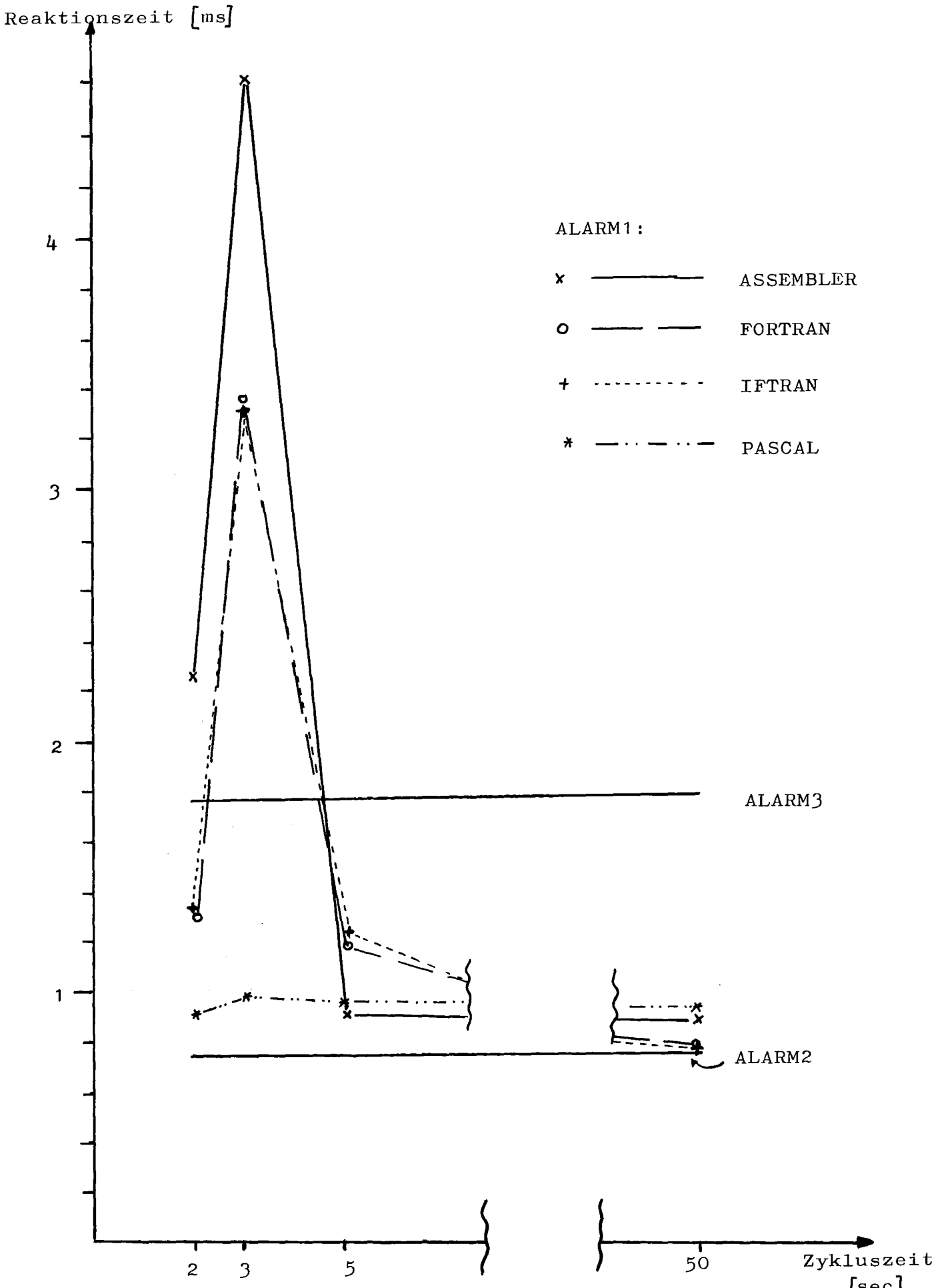


Bild 5.1 Reaktionszeiten für die Tasks ALARM1, 2 und 3

Der konstante Wert von ALARM 3 ist bedingt durch das Simulationssystem und die Treibertasks, die alle eine höhere Priorität als ALARM 3 besitzen.

Sehr unterschiedlich sind die Reaktionszeiten bei ALARM 1. Mit Ausnahme von PASCAL haben alle anderen Programmiersprachen ein relatives Maximum beim 3-sec Zyklus. Es konnte hierbei nicht geklärt werden, durch welchen Umstand diese Spitzenwerte verursacht wurden. Ein Grund dafür ist aber, daß ALARM 1 die niederste Priorität besitzt; alle anderen Tasks werden mit einer höheren Dringlichkeit bearbeitet. Bei kurzen Zykluszeiten kann es nun zu ungünstigen Konstellationen kommen, die die hohen Reaktionszeiten bewirken.

Geht man vom Realbetrieb des Dichtemeßgerätes aus, dessen Zykluszeiten im Bereich zwischen 2 und 5 min liegen, so ergeben sich hier für alle Werte (ab 50 sec Zykluszeiten) nur geringfügig unterschiedliche Reaktionszeiten für alle Tasks.

Die kleinen Zykluszeiten sind nur mit dem Simulationssystem möglich und wurden deshalb durchgeführt, um eine größere Anzahl von Messungen in annehmbarer Zeit ausführen zu können.

5.6 Zeitprofile

Aus den in 5.1 - 5.5 hervorgegangenen Ergebnissen wurde für jede Programmiersprache ein Zeitprofil erstellt, aus dem hervorgeht, welche Programme bzw. Tasks zur eigentlichen Belastung des Rechners am meisten beitragen. Die angegebenen Prozentzahlen sind Durchschnittswerte aus den Tabellen S. 92-95 . Die Zahlen für die Initialisierung ergeben sich aus der Differenz der restlichen Angaben zu der Gesamtbelastung (=100 %).

Zeitprofil ASSEMBLER	ohne VFD
Dispatcher/Clock-Grundbelastung	4.31%
Dispatcher-Overhead	22.04%
Betriebssystemtasks	26.99%
ALARM1 + ALARM3 Rechenzeit	8.19%
ALARM1 + ALARM3 IOC+EXEC	23,70%
ALARM2 Rechenzeit	3.13%
ALARM2 IOC+EXEC	11.34%
Initialisierung	1.25%
Gesamtbelastung / Zyklus	.71 sec

Zeitprofil ASSEMBLER	mit VFD
Dispatcher/Clock-Grundbelastung	4.31%
Dispatcher-Overhead	16,45%
Betriebssystemtasks	57.32%
ALARM1 + ALARM 3 Rechenzeit	4.08%
ALARM1 + ALARM3 IOC+EXEC	12.64%
ALARM2 Rechenzeit	1.06%
ALARM2 IOC+EXEC	4.01%
Initialisierung	.13%
Gesamtbelastung / Zyklus	2.59 sec

Tab. 5.1 Prozentuale Aufteilung der Rechnerbelastung für ASSEMBLER

Zeitprofil	FORTRAN	ohne VFD
Dispatcher/Clock-Grundbelastung		4.31%
Dispatcher-Overhead		19.01%
Betriebssystemtasks		26.43%
ALARM1 + ALARM3 Rechenzeit		14.94%
ALARM1 + ALARM3 IOC+EXEC		18.03%
ALARM2 Rechenzeit		3.27%
ALARM2 IOC+EXEC		11.90%
Initialisierung		2.11%
Gesamtbelastung / Zyklus		.91 sec

Zeitprofil	FORTRAN	mit VFD
Dispatcher/Clock-Grundbelastung		4.31%
Dispatcher-Overhead		15.58%
Betriebssystemtasks		54.79%
ALARM1 + ALARM3 Rechenzeit		8.26%
ALARM1 + ALARM3 IOC+EXEC		10.99%
ALARM2 Rechenzeit		.76%
ALARM2 IOC+EXEC		2.83%
Initialisierung		2.48%
Gesamtbelastung / Zyklus		2.48 sec

Tab. 5.2 Prozentuale Aufteilung der Rechnerbelastung für FORTRAN

Zeitprofil IFTRAN	ohne VFD
Dispatcher/Clock-Grundbelastung	4.31%
Dispatcher-Overhead	19.08%
Betriebssystemtasks	26.33%
ALARM1 + ALARM3 Rechenzeit	15.01%
ALARM1 + ALARM3 IOC+EXEC	18.08%
ALARM2 Rechenzeit	3.22%
ALARM2 IOC+EXEC	11.91%
Initialisierung	1.96%
Gesamtbelastung / Zyklus	.90 sec

Zeitprofil IFTRAN	mit VFD
Dispatcher/Clock-Grundbelastung	4.30%
Dispatcher-Overhead	15.79%
Betriebssystemtasks	54.13%
ALARM1 + ALARM3 Rechenzeit	8.52%
ALARM1 + ALARM3 IOC+EXEC	11.06%
ALARM2 Rechenzeit	.77%
ALARM2 IOC+EXEC	2.89%
Initialisierung	2.54%
Gesamtbelastung / Zyklus	2.43 sec

Tab. 5.3 Prozentuale Aufteilung der Rechnerbelastung für IFTRAN

Zeitprofil	PASCAL	ohne VFD
Dispatcher/Clock-Grundbelastung		4.31%
Dispatcher-Overhead		14.31%
Betriebssystemtasks		21.29%
ALARM1 Rechenzeit		31.06%
ALARM1 IOC+EXEC		15.48%
ALARM2 Rechenzeit		2.47%
ALARM2 IOC+EXEC		8.72%
Initialisierung		2.36%
Gesamtbelastung / Zyklus		1.06 sec

Zeitprofil	PASCAL	mit VFD
Dispatcher/Clock-Grundbelastung		4.30%
Dispatcher-Overhead		6.88%
Betriebssystemtasks		47.30%
ALARM1 Rechenzeit		25.53%
ALARM1 IOC+EXEC		9.85%
ALARM2 Rechenzeit		.70%
ALARM2 IOC+EXEC		2.43%
Initialisierung		3.01%
Gesamtbelastung / Zyklus		2.68 sec

Tab. 5.4 Prozentuale Aufteilung der Rechnerbelastung für PASCAL

Aus diesen Zeitprofilen kann man ablesen, daß 40 - 70 % der Gesamtbelastung auf Betriebssystemtasks und die Taskverwaltung entfallen. Trennt man innerhalb der Benutzertask die Zeiten für EXEC und IOC ab, auf deren Programmierung der Benutzer keinen Einfluß hat, so nimmt die Belastung durch das Betriebssystem auf 75 - 95 % (!) zu. Eine eventuelle Optimierung der Benutzerprogramme hat, wie aus den Untersuchungen hervorgeht, nur einen geringen Einfluß auf die Gesamtbelastung des Rechners. Vielmehr erscheint es aufgrund dieser Tatsachen erforderlich, im Betriebssystem entsprechende Verbesserungen vorzunehmen.

Betrachtet man abschließend die Gesamtbelastung/Zyklus, so sind bei fehlender Ausgabe auf den halbgraphischen Bildschirm FORTRAN und IFTRAN nur etwa den Faktor 1.3 langsamer. Werden zusätzlich noch Daten auf VFD ausgegeben, so ist zwischen allen Programmiersprachen kein nennenswerter Unterschied zu bemerken.

	ASSEMBLER	FORTRAN	IFTRAN	PASCAL
mit VFD	1	1.28	1.27	1.49
ohne VFD	1	.96	.94	1.03

Verhältnis der Gesamtbelastung/Zyklus zu Assembler

Die Werte < 1 für FORTRAN und IFTRAN sind auf die größeren IOC-Zeiten bei ASSEMBLER zurückzuführen. Diese Zeiten müßten eigentlich für alle Sprachen gleich sein; gewisse Schwankungen sind aber dadurch möglich, daß nicht alle Prozeßüberwachungssysteme exakt gleich sind.

5.7 Bewertung der Sprachen hinsichtlich der Zeitmessungen

Unter den in 5.6 aufgestellten Tatsachen scheint es zunächst unerheblich, welche Programmiersprache man verwendet, da die wesentliche Belastung des Rechners durch Betriebssystemkomponenten hervorgerufen wird. Der große Unterschied zwischen den Rechenzeiten aus 5.2 und 5.3 wird durch den prozentualen sehr großen Anteil von Betriebssystemroutinen stark verringert. Dabei ist aber zu beachten, daß dies nur durch die intensive EA-Tätigkeit von Prozeßprogrammen bewirkt wird.

Aus diesen Gründen besteht eigentlich kein Anlaß, die höheren Programmiersprachen nicht bei der Prozeßprogrammierung einzusetzen, zumal der Unterschied von FORTRAN/IFTRAN zu ASSEMBLER nur gering ist. Bei PASCAL ist der Abstand schon etwas größer, was aber für einen grundsätzlichen Ausschluß dieser Sprache nicht ausreicht.

Die Verwendung der einzelnen Sprachen dürfte aber auch sehr stark vom jeweiligen Problem abhängen. Sind sehr schnelle Reaktionen auf Ereignisse notwendig, so wird zweifellos ASSEMBLER vorzuziehen sein; bei dem weitaus größeren Teil von Aufgabenstellungen aus der Prozeßprogrammierung besteht aber kein Grund, die höheren Programmiersprachen FORTRAN/IFTRAN und PASCAL nicht zu verwenden.

6. Schlußbeurteilung

Abschließend soll eine Bewertung der einzelnen Sprachen im Gesamtzusammenhang dieser Diplomarbeit abgegeben werden.

Beurteilt man die Programmiersprachen hinsichtlich der Möglichkeit, die vom Betriebssystem angebotenen Dienstleistungen einsetzen zu können, so bieten hier neben ASSEMBLER FORTRAN und IFTRAN die besten Voraussetzungen. PASCAL unterstützt keine Intertaskkommunikation und die EA-Fähigkeiten sind stark eingeschränkt. Außerdem ist es mit der gegenwärtigen Version nicht möglich, mehrere PASCAL-Tasks im Multitaskingbetrieb nebeneinander auszuführen. Aus diesen Gründen sollte PASCAL in diesem Einsatzgebiet nicht verwendet werden. BASIC scheidet vorweg aus der Betrachtung aus, da die Programme nur im Background laufen und neben anderen schwerwiegenden Beschränkungen ebenfalls kein Multitasking möglich ist.
Nimmt man die aus den Zeitmessungen gewonnenen Ergebnisse mit hinzu, so sind die Programmiersprachen IFTRAN, FORTRAN und ASSEMBLER am besten dafür geeignet, Prozeßprogramme zu erstellen.
Noch einmal soll aber der starke Verwaltungsaufwand des Betriebssystems erwähnt werden, der sich in sehr großen Rechenzeiten für Systemprogramme bemerkbar macht. Effizienzbetrachtungen sind also nicht nur auf Benutzerprogramme anzuwenden. Wie aus den Messungen hervorgeht, könnten Optimierungen innerhalb des Betriebssystems stärker zu einem besseren Verhalten führen.

Die in dieser Arbeit gemachten Ergebnisse dürfen nicht isoliert gesehen werden. Zu einer umfassenden Beurteilung sind auch die in /1/ festgestellten Eigenschaften der verschiedenen Programmiersprachen heranzuziehen.

Literatur

/1/ M. Swieczkowski
Vergleich verschiedener Sprachen für die Prozeßprogrammierung anhand der Mehrfachimplementierung eines Überwachungssystems
KfK - Bericht 2714 B (Januar 1979)

/2/ Varian Data Machines
VORTEX II Operating System Reference Manual
Best.-Nr. 98 A 9952 244

/3/ H. Borrmann
Programmabwicklung im Rechner-Betriebssystem VORTEX I
KFK - Ext 13/75-4

/4/ K. Landmark
V$CMA, ein Programm zum Austausch von Meldungen zwischen verschiedenen Tasks unter dem VORTEX-Betriebssystem
(1977) unveröffentlicht

/5/ H. Borrmann
Ein/Ausgabe im Rechner-Betriebssystem VORTEX I/II
KFK - Ext 13/75-3

/6/ Varian Data Machines
Memory Map Manual
Best.-Nr. 98 A 9906 100

/7/ Varian Data Machines
FORTRAN IV Reference Manual
Best.-Nr. 98 A 9952 040

/8/ Grauer, Didic
MECROS - Ein System zur Messung von Leistungskenngrößen des Realzeitbetriebssystems CALAS 70
KFK - Bericht 2270 März 1976

/9/ Varian Data Machines
PASCAL Reference Manual
Best.-Nr. 81 A 0411-001 A

/10/ General Research Corporation
IFTRAN: Structured Programming Preprocessors for FORTRAN
Santa Barbara, California

/11/ Varian Data Machines
VARIAN Assembly Reference Manual
Best.-Nr. 98 A 9952 450

/12/ Varian Data Machines
V70/620 BASIC Language
Best.-Nr. 98 A 9952-033

/13/ Kapp
Das Pascal-System der Varian V75
(1977) unveröffentlicht

/14/ H. Borrmann
Programmsystem zur zentralen Prozeßkommunikation über ein MOPEK-Interface im Rahmen der VORTEX-Organisation
KFK - Ext 13/77-1

/15/ Kroenig, D.
Praxis von Sprachen, Programmiersystemen und Programmgeneratoren
1. Treffen 76 des German Chapter of the ACM
Applied computer Science 3
(Berichte zur praktischen Informatik)
Carl Hanser Verlag München 1976

/16/ W. Rosenbohm
Ein Software-Meßsystem für Prozeßrechner
Informatik Fachberichte 7
Fachtagung Prozeßrechner 1977
Springer Verlag 1977 pp 513

/17/ W. R. Deniston
SIPE: A TSS/360 Software Measurement Technique
Proc. ACM 24th Nat. Conference 1969 pp 229

/18/ Cantrell, Ellison
Multiprogramming system performance measurement and analysis
AFIPS Spring Joint Comp. Conf. 1968 pp 213

/19/ Koster, Bussell
Instrumenting computer systems and their programs
AFIPS Fall Joint Comp. Conf. 1972 pp 525

/20/ Lyon, Stillman
Simple Transforms for Instrumenting FORTRAN-Decks
Software - Practice and Experience Vol 5 1975
pp 347

Anhang A

Meßwerte für Kapitel 5

Erläuterung:

Systemrechenzeit = Betriebssystemtasks + Gesamtrechenzeit
für Benutzertasks + Dispatcher-Overhead

Gesamtbelastung = Systemrechenzeit + Grundbelastung
für Dispatcher/Clock

	ASSEMBLER	FORTRAN/IFTRAN	PASCAL
Zeit für einen Aufruf (in ms)	.87	2.23	4.87
Verhältnis zu ASSEMBLER	1	2.55	5.58

Rechenzeiten für das Unterprogramm MDRECH (Meßwertanpassung)

	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
ASSEMBLER	2.41	4.93	2.26	4.61	2.26	4.50	1.05	2.13	3.17
FORTRAN	5.40	11.06	5.41	11.02	5.26	10.68	2.08	4.07	6.07
IFTRAN	5.40	10.71	5.46	11.12	5.31	10.79	2.11	4.11	6.14
PASCAL	13.09	25.77	13.40	26.80	13.07	26.24	7.36	13.38	19.48
ASSEMBLER	1	1	1	1	1	1	1	1	1
FORTRAN	2.24	2.24	2.39	2.39	2.32	2.32	1.98	1.91	1.91
IFTRAN	2.24	2.17	2.42	2.41	2.35	2.35	2.01	1.93	1.94
PASCAL	5.433	5.23	5.93	5.81	5.78	5.70	7.01	6.28	6.14

Zeile 1 - 4 : absolut Werte in sec
Zeile 5 - 8 : Verhältnis zu ASSEMBLER

Programmrechenzeiten von ALARM1
und ALARM3

	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
ASSEMBLER	5.67	11.60	5.35	10.93	5.28	10.74	2.91	5.84	8.72
FORTRAN	5.30	10.96	5.37	10.98	5.14	10.48	2.45	4.86	7.27
IFTRAN	5.23	10.34	5.36	10.97	5.14	10.50	2.42	4.79	7.16
PASCAL	5.33	10.47	5.51	10.99	5.26	10.49	2.37	4.66	6.98
ASSEMBLER	1	1	1	1	1	1	1	1	1
FORTRAN	.93	.94	1.00	1.00	.97	.98	.84	.83	.83
IFTRAN	.92	.89	1.00	1.00	.97	.98	.83	.82	.82
PASCAL	.94	.90	1.03	1.01	1.00	.98	.81	.80	.80

IOC - Zeit von ALARM1 + ALARM3

Zeile 1 - 4 : absolute WERTE in sec
Zeile 5 - 8 : Verhältnis zu ASSEMBLER

	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
ASSEMBLER	1.30	2.65	1.22	2.50	1.20	2.42	.36	.72	1.07
FORTRAN	1.19	2.47	1.21	2.47	1.19	2.36	.29	.57	.85
IFTRAN	1.18	2.32	1.22	2.49	1.16	2.36	.29	.57	.86
PASCAL	1.21	2.37	1.25	2.49	1.20	2.38	.30	.58	.86
ASSEMBLER	1	1	1	1	1	1	1	1	1
FORTRAN	.92	.93	.99	.99	.99	.98	.81	.79	.79
IFTRAN	.91	.88	1.00	1.00	.97	.98	.81	.79	.80
PASCAL	.93	.89	1.02	1.00	1.00	.98	.83	.83	.80

EXEC - Zeit von ALARM1 + ALARM3

Zeile 1 - 4 : absolute Werte in sec
Zeile 5 - 8 : Verhältnis zu ASSEMBLER

ASSEMBLER	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
Rechenzeit	2.41	4.93	2.26	4.61	2.26	4.60	1.05	2.13	3.17
IOC	5.67	11.60	5.35	10.93	5.28	10.74	2.91	5.84	8.72
EXEC	1.30	2.65	1.22	2.50	1.20	2.42	.36	.72	1.07
①	.35	.35	.34	.34	.35	.35	.32	.32	.32
FORTRAN									
Rechenzeit	5.40	11.06	5.41	11.02	5.26	10.68	2.08	4.07	6.07
IOC	5.30	10.96	5.37	10.98	5.14	10.48	2.45	4.86	7.27
EXEC	1.19	2.47	1.21	2.47	1.16	2.36	.29	.57	.85
①	.83	.82	.82	.82	.84	.83	.76	.75	.75

1 Verhältnis $\frac{\text{Programmrechenzeit}}{\text{IOC + EXEC}}$

absolute Programmrechenzeit, IOC- und EXEC-Zeit zusammen für die Tasks ALARM1 + ALARM3 (in sec) und Verhältnis von Programmrechenzeit zu (IOC+EXEC)-Zeit für ASSEMBLER und FORTRAN

IFTRAN	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
Rechenzeit	5.40	10.71	5.46	11.12	5.31	10.79	2.11	4.11	6.14
IOC	5.23	10.34	5.36	10.97	5.14	10.50	2.42	4.79	7.16
EXEC	1.18	2.32	1.22	2.49	1.16	2.36	.29	.57	.86
①	.84	.85	.83	.83	.84	.84	.78	.77	.77
PASCAL									
Rechenzeit	13.09	25.77	13.40	26.80	13.07	26.24	7.36	13.38	19.48
IOC	5.33	10.47	5.51	10.99	5.26	10.49	2.37	4.66	6.98
EXEC	1.21	2.37	1.25	2.49	1.20	2.38	.30	.58	.86
①	2.00	2.01	1.98	1.99	2.02	2.04	2.75	2.55	2.48

1 Verhältnis $\frac{\text{Programmrechenzeit}}{\text{IOC + EXEC}}$

absolute Programmrechenzeit, IOC- und EXEC-Zeit zusammen für die Tasks ALARM1 + ALARM3 (bei PASCAL nur ALARM1) (in sec) und Verhältnis von Programmrechenzeit zu (IOC+EXEC)-Zeit für IFTRAN und PASCAL

	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
ASSEMBLER	9.38	19.18	8.83	18.04	8.74	17.76	4.32	8.69	12.96
FORTRAN	11.89	24.49	11.99	24.47	11.59	23.52	4.82	9.50	14.19
IFTRAN	11.81	23.37	12.04	24.58	11.61	23.65	4.82	9.47	14.16
PASCAL	19.63	38.61	20.16	40.28	19.53	39.11	10.03	18.62	27.32
ASSEMBLER	1	1	1	1	1	1	1	1	1
FORTRAN	1.27	1.28	1.36	1.36	1.33	1.33	1.12	1.09	1.09
IFTRAN	1.26	1.22	1.36	1.36	1.33	1.33	1.12	1.09	1.09
PASCAL	2.09	2.01	2.28	2.32	2.23	2.20	2.32	2.14	2.11

Gesamtrechenzeit von ALARM1 + ALARM3

Zeile 1 - 4 : absolute Werte in sec
Zeile 5 - 8 : Verhältnis zu ASSEMBLER

	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
ASSEMBLER	14.05	28.68	12.86	26.56	12.48	25.35	5.72	11.41	16.78
FORTRAN	19.42	38.66	18.41	36.39	17.20	34.13	6.73	12.30	17.89
IFTRAN	19.30	37.43	18.44	36.47	17.25	34.12	6.75	12.28	17.85
PASCAL	26.54	51.06	26.06	50.66	25.10	48.83	12.22	21.63	31.17
ASSEMBLER	1	1	1	1	1	1	1	1	1
FORTRAN	1.38	1.35	1.43	1.37	1.38	1.35	1.18	1.08	1.07
IFTRAN	1.37	1.31	1.43	1.37	1.38	1.35	1.18	1.08	1.06
PASCAL	1.89	1.78	2.02	1.91	2.01	1.93	2.14	1.90	1.86

Zeile 1 - 4 : absolute Werte in sec
Zeile 5 - 8 : Verhältnis zu ASSEMBLER

Gesamtrechenzeit für alle Benutzertasks

	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
ASSEMBLER	8.17	16.02	7.65	15.06	7.42	14.59	15.03	29.46	44.29
FORTRAN	10.47	19.49	10.01	18.22	9.54	17.35	14.48	26.57	39.04
IFTRAN	10.42	19.04	9.90	18.26	9.48	17.31	13.84	26.07	37.95
PASCAL	9.84	17.71	9.49	17.17	9.32	16.34	13.40	24.91	36.53
ASSEMBLER	1	1	1	1	1	1	1	1	1
FORTRAN	1.28	1.22	1.31	1.21	1.29	1.19	.96	.90	.88
IFTRAN	1.28	1.19	1.29	1.21	1.28	1.19	.92	.88	.86
PASCAL	1.20	1.11	1,24	1.14	1.26	1.12	.89	.85	.82

Ausführungszeiten für Betriebssystemtasks

Zeile 1 - 4 : absolute Werte in sec
Zeile 5 - 8 : Verhältnis zu ASSEMBLER

	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
ASSEMBLER	6.63	12.67	6.31	12.17	6.23	12.01	4.49	8.43	12.24
FORTRAN	7.35	14.36	6.86	13.34	6.75	13.12	4.12	7.59	11.05
IFTRAN	7.38	13.93	6.92	13.40	6.79	13.16	4.14	7.51	10.94
PASCAL	6.58	12.35	6.37	11.97	5.85	10.98	2.18	3.51	4.88
ASSEMBLER	1	1	1	1	1	1	1	1	1
FORTRAN	1.11	1.13	1.09	1.10	1.08	1.09	.92	.90	.90
IFTRAN	1.11	1.10	1.10	1.10	1.09	1.10	.92	.89	.89
PASCAL	.99	.97	1.01	.98	.94	.91	.49	.42	.40

Dispatcher-Overhead

Zeile 1 - 4 : absolute Werte in sec
Zeile 5 - 8 : Verhältnis zu ASSEMBLER

	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
ASSEMBLER	1.72	1.79	1.68	1.76	1.68	1.74	.38	.39	.38
FORTRAN	1.85	1.98	1.84	2.00	1.80	1.97	.46	.46	.46
IFTRAN	1.85	1.97	1.86	2.00	1.82	1.97	.49	.47	.47
PASCAL	2.70	2.88	2.74	2.95	2.69	2.99	.91	.87	.85
ASSEMBLER	2.12	2.26	2.04	2.18	2.00	2.11	1.27	1.35	1.37
FORTRAN	2.64	2.69	2.68	2.73	2.55	2.60	1.63	1.62	1.62
IFTRAN	2.62	2.69	2.66	2.72	2.54	2.59	1.63	1.64	1.63
PASCAL	4.03	4.13	4.08	4.23	4.29	4.45	5.61	6.16	6.39

Zeile 1 - 4 : $\frac{\text{Gesamtrechenzeit für alle Benutzertasks}}{\text{Rechenzeit f. Systemtasks}}$

Zeile 5 - 8 : $\frac{\text{Gesamtrechenzeit für alle Benutzertasks}}{\text{Dispatcher-Overhead}}$

ASSEMBLER	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
ALARM1	2.23	2.23	4.59	4.61	.88	.88	.83	.84	.86
ALARM3	1.75	1.75	1.75	1.75	1.75	1.75	1.75	1.75	1.75
ALARM2	.72	.71	.73	.73	.72	.72	.72	.73	.73
FORTRAN									
ALARM1	1.27	1.28	3.35	3.37	1.19	1.20	.79	.79	.79
ALARM3	1.75	1.75	1.75	1.75	1.75	1.75	1.75	1.75	1.75
ALARM2	.73	.72	.73	.73	.73	.72	.74	.73	.73
IFTRAN									
ALARM1	1.31	1.29	3.30	3.34	1.23	1.24	.79	.80	.80
ALARM3	1.75	1.75	1.75	1.75	1.75	1.75	1.75	1.75	1.75
ALARM2	.72	.73	.72	.72	.73	.73	.72	.72	.72
PASCAL									
ALARM1	.90	.90	1.03	.89	.89	.93	.93	.86	.91
ALARM2	.72	.72	.72	.72	.72	.73	.71	.73	.73

Durchschnittliche Reaktionszeiten auf Interrupts [in ms]

ASSEMBLER	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
Gesamt-Rechenzeit	28.85	57.37	26.82	53.79	26.13	51.95	25.24	49.30	73.31
DC-Grundbelast.	1.30	2.58	1.21	2.42	1.18	2.34	1.14	2.22	3.29
Dispatcher Overhead	6.63	12.67	6.31	12.17	6.23	12.01	4.49	8.43	12.24
Betriebs-systemtasks	8.17	16.02	7.65	15.06	7.42	14.59	15.03	29.46	44.29
A1 + A3 Rechenzeit	2.41	4.93	2.26	4.61	2.26	4.60	1.05	2.13	3.17
A1 + A3 IOC + EXEC	6.97	14.25	6.57	13.43	6.48	13.16	3.27	6.56	9.79
A2 Rechenzeit	.99	2.02	.85	1.75	.81	1.64	.28	.56	.80
A2 IOC + EXEC	3.57	7.29	3.14	6.42	2.89	5.89	1.07	2.10	2.98

Verteilung der Rechnerbelastung auf einzelne Programmteile für ASSEMBLER — absolute Zeiten in sec

DC = Dispatcher-Clock

A = ALARM

FORTRAN	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
Gesamt-Rechenzeit	37.24	72.51	35.01	67.95	33.49	64.60	25.33	46.46	67.98
DC-Grundbelast.	1.68	3.26	1.58	3.06	1.51	2.91	1.14	2.09	3.06
Dispatcher-Overhead	7.35	14.36	6.86	13.34	6.75	13.12	4.12	7.59	11.05
Betriebs-systemtasks	10.47	19.49	10.01	18.22	9.54	17.35	14.48	26.57	39.04
A1 + A3 Rechenzeit	5.40	11.06	5.41	11.02	5.26	10.68	2.08	4.07	6.07
A1 + A3 IOC + EXEC	6.40	13.43	6.58	13.45	6.33	12.84	2.74	5.43	8.12
A2 Rechenzeit	1.40	2.84	1.14	2.31	.99	2.05	.19	.38	.56
A2 IOC + EXEC	5.09	10.29	4.17	8.43	3.62	7.52	.70	1.40	2.10

Verteilung der Rechnerbelastung auf einzelne Programmteile für FORTRAN — absolute Zeiten in sec

DC = Dispatcher-Clock

A = ALARM

IFTRAN	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
Gesamt-Rechenzeit	37.10	70.40	35.26	68.13	33.52	64.59	24.73	45.86	66.74
DC-Grundbelast.	1.67	3.17	1.59	3.07	1.51	2.91	1.11	2.06	3.00
Dispatcher-Overhead	7.38	13.93	6.92	13.40	6.79	13.16	4.14	7.51	10.94
Betriebs-systemtasks	10.42	19.04	9.90	18.26	9.48	17.31	13.84	26.07	37.95
A1 + A3 Rechenzeit	5.40	10.71	5.46	11.12	5.31	10.79	2.11	4.11	6.14
A1 + A3 IOC + EXEC	6.41	12.66	6.58	13.46	6.30	12.86	2.71	5.36	8,02
A2 Rechenzeit	1.38	2.79	1.12	2.28	.97	1.99	.19	.37	.55
A2 IOC + EXEC	5.07	10.22	4.16	8.41	3.63	7.43	.71	1.40	2.10

Verteilung der Rechnerbelastung auf einzelne Programmteile für IFTRAN — absolute Zeiten in sec

DC = Dispatcher-Clock

A = ALARM

PASCAL	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
Gesamt-Rechenzeit	42.96	81.12	41.88	79.80	40.27	76.15	27.80	50.05	72.58
DC-Grundbelast.	1.93	3.65	1.89	3.59	1.81	3.43	1.25	2.25	3.27
Dispatcher-Overhead	6.58	12.35	6.37	11.97	5.85	10.98	2.18	3.51	4.88
Betriebs-systemtasks	9.84	17.71	9.49	17.17	9.32	16.34	13.40	24.91	36.53
A1 Rechenzeit	13.09	25.77	13.40	26.80	13.07	26.24	7.36	13.38	19.48
A1 IOC + EXEC	6.54	12.84	6.76	13.48	6.46	12.87	2.67	5.24	7.84
A2 Rechenzeit	1.22	2.44	.99	1.99	.93	1.86	.19	.37	.55
A2 IOC + EXEC	4.34	8.65	3.51	7.04	3.29	6.51	.66	1.29	1.94

Verteilung der Rechnerbelastung auf einzelne Programmteile für PASCAL — absolute Zeiten in sec

DC = Dispatcher-Clock

A = ALARM

ASSEMBLER	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
Gesamt-belastung	30.15	59.95	28.03	56.21	27.31	54.29	26.38	51.52	76.60
DC Grundbelast.	4.31	4.30	4.32	4.31	4.32	4.31	4.32	4.31	4.30
Dispatcher-Overhead	21.99	21.13	22.51	21.65	22.81	22.12	17.02	16.36	15.98
Betriebs-systemtasks	27.10	26.72	27.29	26.79	27.17	26.87	56.97	57.18	57.82
A1 + A3 Rechenzeit	7.99	8.22	8.06	8.20	8.28	8.47	3 98	4.13	4.13
A1 + A3 IOC + EXEC	23.12	23.77	23.44	23.89	23.73	24.24	12.40	12.73	12.78
A2 Rechenzeit	3.28	3.37	3.03	3.11	2.97	3.02	1.06	1.09	1.04
A2 IOC + EXEC	11.84	12.16	11.20	11.42	10.58	10.85	4.06	4.08	3.89

Prozentuale Verteilung der Rechnerbelastung für ASSEMBLER

Zeile 2 - 8

Zeile 1 absolute Zeit in sec

FORTRAN	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
Gesamt-belastung	38.92	75.77	36.59	71.01	35.00	67.51	26.47	48.55	71.04
DC-Grundbelast.	4.32	4.30	4.32	4.31	4.31	4.31	4.31	4.30	4.31
Dispatcher-Overhead	18.88	18.95	18.75	18.79	19.28	19.43	15.56	15.53	15.55
Betriebs-systemtasks	26.90	25.72	27.36	25.66	27.26	25.70	54.70	54.73	54.95
A1 + A3 Rechenzeit	13.87	14.60	14.79	15.52	15.03	15.82	7.86	8.38	8.54
A1 + A3 IOC + EXEC	16.44	17.72	17.98	18.94	18.09	19.02	10.35	11.18	11.43
A2 Rechenzeit	3.60	3.75	3.12	3.25	2.83	3.04	.72	.78	.79
A2 IOC + EXEC	13.08	13.58	11.40	11.87	10.34	11.14	2.64	2.88	2.96

Prozentuale Verteilung der Rechnerbelastung für FORTRAN

Zeile 2 - 8

Zeile 1 absolute Zeit in sec

IFTRAN	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
Gesamt-belastung	38.77	73.57	36.85	71.20	35.03	67.50	25.84	47.92	69.74
DC-Grundbelast.	4.31	4.31	4.31	4.31	4.31	4.31	4.30	4.30	4.30
Dispatcher-Overhead	19.04	18.93	18.78	18.82	19.38	19.50	16.02	15.67	15.69
Betriebs-systemtasks	26.88	25,88	26.86	25.65	27.06	25.64	53.56	54.40	54.42
A1 + A3 Rechenzeit	13.93	14,56	14.82	15.62	15.16	15.99	8.17	8.58	8.80
A1 + A3 IOC + EXEC	16.53	17.21	18.86	18.90	17.98	19.05	10.49	11.19	11.50
A2 Rechenzeit	3.56	3.79	3.04	3.20	2.77	2.95	.74	.77	.79
A2 IOC + EXEC	13.08	13.89	11.29	11.81	10.36	11.01	2.75	2.92	3.01

Prozentuale Verteilung der Rechnerbelastung für IFTRAN

Zeile 2 - 8

Zeile 1 absolute Zeit in sec

PASCAL	40/2	80/2	40/3	80/3	40/5	80/5	10/50	20/50	30/50
Gesamt-belastung	44.89	84.77	43.77	83.39	42.08	79.58	29.05	52.30	75.85
DC-Grundbelast.	4.30	4.31	4.32	4.31	4.30	4.31	4.30	4.30	4.31
Dispatcher-Overhead	14.66	14.57	14.55	14.35	13.90	13.80	7.50	6.71	6.43
Betriebs-systemtasks	21.92	20.89	21.68	20.59	22.15	20.53	46.13	47.62	48.16
A1 Rechenzeit	29.16	30.40	30.61	32.14	31.06	32.97	25.34	25.58	25.68
A1 IOC + EXEC	14.57	15.15	15.44	16.17	15.35	16.17	9.19	10.02	10.34
A2 Rechenzeit	2.72	2.88	2.26	2.39	2.21	2.34	.65	.71	.73
A2 IOC + EXEC	9.67	10.20	8.02	8.44	7.81	8.18	2.27	2.47	2.56

Prozentuale Verteilung der Rechnerbelastung für PASCAL

Zeile 2 - 8

Zeile 1 absolute Zeit in sec

Anhang B

Programmlistings für MDRECH

```
 1          TITLE    MDRECH
 2          NAME     MDRECH
 3
 4 X        EQU      1
 5 TWO      EQU      0422
 6 FIVE     EQU      0465
 7 NINE     EQU      0470
 8
 9          EXT      $SE
10          EXT      $PC              INTEGER -> REAL
11          EXT      $IC              REAL -> INTEGER
12
13 FAD      MAC                       FLOATING PONT ADD
14          DATA     0105134,P(1)
15          EMAC
16 FMU      MAC                       FLOATING POINT MULTIPLY
17          DATA     0105074,P(1)
18          EMAC
19 FDV      MAC                       FLOATING POINT DIVIDE
20          DATA     0105034,P(1)
21          EMAC
22 FLD      MAC                       LOAD AB
23          DATA     0105032,P(1)
24          EMAC
25 FST      MAC                       STORE AB
26          DATA     0105033,P(1)
27          EMAC
28
29 MDRECH   ENTR
30          CALL     $SE,4
31 PAR      BSS      4
32          STA      AA
33          STX      XX
34          TZX
35          STX      FERTIG
36          STX      FER
37          STX      FER+1
38          LDA      REAL1
39          CALL     $PC
40          FST      F1
41          LDA      REAL2
42          CALL     $PC
43          FST      F2
44          LDA      FAKTOR
45          CALL     $PC
46          FST      F3
47 *
48 *    DIE MESSWERTE WERDEN UEBER WCS
49 *    VOM MUPEK-WERTEBEREICH IN DEN WERTEBEREICH
50 *    DES MESSGERAETES UMGESETZT
51 *
52 LOOP1    LDAE* PAR,X
53          CALL     $PC
54          FAD      F1
55          FDV      F2
56          FMU      F3
57          CALL     $IC
58          STAE     AW,X
59          IXR
```

Unterprogramm MDRECH (Meßwertanpassung) für ASSEMBLER

```
 60          TXA
 61          SUB       TWO
 62          JAN       LOOP1
 63          LDA       AW+1            UEBERGABE IN PARAMETERFELD
 64          STAE*     PAR+1           FUER AUSGABE:  AW2
 65          LDA       AW
 66          STAE*     PAR                            AW1
 67          SUB       AW+1
 68          STAE*     PAR+2                          DIFF
 69          TZX
 70 LOOP2    LDAE      POOL,X          IST DER NEUE MESSWERT
 71          SUB       NINE            INNERHALB EINER DELTA-
 72          SUBE      AW,X            UMGEBUNG DES ZUVOR ERHALTENEN
 73          JAP       L021            WERTES:
 74          LDAE      POOL,X          POOL-8 <= AW <= POOL+8 ?
 75          ADD       NINE
 76          SUBE      AW,X
 77          JAP       L022
 78 L021     TZA
 79          STAE      INDEX,X
 80 L022     INRE      INDEX,X
 81          LDAE      AW,X            RETTEN DES NEUEN MESSWERTES
 82          STAE      POOL,X
 83          LDAE      INDEX,X         SIND SCHON 5 "GUTE"
 84          SUB       FIVE            MESSWERTE ERFASST WORDEN?
 85          JAN       L023            NEIN:
 86          INRE      FER,X           JA:
 87 L023     IXR
 88          TXA
 89          SUB       TWO
 90          JAN       LOOP2
 91          LDA       FER
 92          ANA       FER+1
 93          JAZ       ENDE
 94          INRE      FERTIG
 95 ENDE     LDA       FERTIG
 96          STAE*     PAR+3
 97          LDA       AA
 98          LDX       XX
 99          RETU*     MDRECH
100          COMN      15
101 POOL     COMN      2
102 INDEX    COMN      2
103          COMN      3
104 STEIG    COMN      1
105 ORDNTE   COMN      1
106 AA       DATA      0
107 XX       DATA      0
108 AWF      DATA      0
109 AW       DATA      0,0
110 FER      DATA      0,0
111 FERTIG   DATA      0
112 REAL1    DATA      3075
113 REAL2    DATA      4100
114 FAKTOR   DATA      10000
115 F1       BSS       2
116 F2       BSS       2
117 F3       BSS       2
118          END
```

Unterprogramm MDRECH (Meßwertanpassung) für ASSEMBLER
(Fortsetzung)

```
 1        SUBROUTINE MDRECH(AW1,AW2,DIFF,FERTIG)
 2        INTEGER AW1,AW2,DIFF,FAKTOR
 3        INTEGER IVZG(11),TMEBUF(4),POOL(2,1),INDEX(2),
 4       *         SIMRE,AV,ZYKAN7,STEIG,ORDNTE,AUSVAR
 5        LOGICAL FERTIG,F1,F2
 6        COMMON IVZG,TMEBUF,POOL,INDEX,
 7       *        SIMRE,AV,ZYKAN7,STEIG,ORDNTE,AUSVAR
 8        FAKTOR=10000
 9        FERTIG=.FALSE.
10        F1=.FALSE.
11        F2=.FALSE.
12        AW1=IFIX((FLOAT(AW1)+3075.)/4100.*FLOAT(FAKTOR))
13        AW2=IFIX((FLOAT(AW2)+3075.)/4100.*FLOAT(FAKTOR))
14        DIFF=AW1-AW2
15        IF (AW1.LT.(POOL(1,1)-8).OR.AW1.GT.(POOL(1,1)+8)) INDEX(1)=0
16        INDEX(1)=INDEX(1)+1
17        POOL(1,1)=AW1
18        IF (INDEX(1).GE.5) F1=.TRUE.
19        IF (AW2.LT.(POOL(2,1)-8).OR.AW2.GT.(POOL(2,1)+8)) INDEX(2)=0
20        INDEX(2)=INDEX(2)+1
21        POOL(2,1)=AW2
22        IF (INDEX(2).GE.5) F2=.TRUE.
23        IF (F1.AND.F2) FERTIG=.TRUE.
24        RETURN
25        END
```

Unterprogramm MDRECH (Meßwertanpassung) für FORTRAN und IFTRAN

```
 1 PROCEDURE MDRECH(VAR AW1,AW2,DIFF:INTEGER;
 2                   VAR FERTIG:BOOLEAN);
 3 CONST RE111=3075.0;     RE222=4100.0;
 4 VAR F1,F2:BOOLEAN;
 5 BEGIN
 6    FERTIG:=FALSE;
 7    F1:=FALSE;
 8    F2:=FALSE;
 9    AW1:=TRUNC((CONV(AW1)+RE111)/RE222*CONV(FAKTOR));
10    AW2:=TRUNC((CONV(AW2)+RE111)/RE222*CONV(FAKTOR));
11    DIFF:=AW1-AW2;
12    IF (AW1<(POOL(.1.)-8)) OR (AW1>(POOL(.1.)+8)) THEN INDEX(.1.):=0;
13    INDEX(.1.):=INDEX(.1.)+1;
14    POOL(.1.):=AW1;
15    IF INDEX(.1.)>=5 THEN F1:=TRUE;
16    IF (AW2<(POOL(.2.)-8)) OR (AW2>(POOL(.2.)+8)) THEN INDEX(.2.):=0;
17    INDEX(.2.):=INDEX(.2.)+1;
18    POOL(.2.):=AW2;
19    IF INDEX(.2.)>=5 THEN F2:=TRUE;
20    IF F1 AND F2 THEN FERTIG:=TRUE
21 END;   "MDRECH"
```

Unterprogramm MDRECH (Meßwertanpassung) für PASCAL