KfK 2642
April 1978

Tagungsbericht der 9. Jahrestagung des Siemens Prozeßrechner Anwenderkreises I vom 5. bis 7. April 1978 im Kernforschungszentrum Karlsruhe

Herausgeber:
U. Voges
Institut für Datenverarbeitung in der Technik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Datenverarbeitung in der Technik

KfK 2642

Tagungsbericht

der

9. Jahrestagung des Siemens Prozeßrechner Anwenderkreises I

vom 5. bis 7. April 1978

im Kernforschungszentrum Karlsruhe

herausgegeben von

Udo Voges

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Zusammenfassung

Auf der 9. Jahrestagung des Siemens Prozeßrechner Anwenderkreises wurde in 27 Vorträgen über die Anwendung und den Einsatz von Siemens Prozeßrechnern in verschiedenen Bereichen berichtet. Dabei wurden sowohl Hardware- wie auch Software-Entwicklungen vorgestellt. Außerdem wurde über einige geplante Entwicklungen vorgetragen.

Proceedings of the 9th Annual Meeting of the Siemens Processcomputer Users Group, April 5-7, 1978 in the Nuclear Research Center Karlsruhe

Abstract

These Proceedings contain the papers presented at the 9th Annual Meeting of the Siemens Processcomputer Users Group. The papers cover different application areas of minicomputers, like computer nets, data base systems, compilers, picture processing etc. Not only software but also hardware aspects are dealt with. Some papers report on future trends.

Inhaltsverzeichnis

Vorwort	1
Programm	3
Vorträge (Seitenangaben siehe Programm)	7
Besichtigungen während der Jahrestagung	302
Teilnehmerverzeichnis	303
Protokoll der SAK-Jahresversammlung	308
Adressen des SAK-Vorstandes	310

Vorwort

Vom 5. bis 7. April 1978 fand im Kernforschungszentrum Karlsruhe die 9. Jahrestagung des Siemens Prozeßrechner Anwenderkreises I (SAK I) statt, an der 118 Anwender und Mitarbeiter von Ausbildungs- und Forschungsstätten sowie von Firmen, hier insbesondere von Siemens, teilnahmen.

Auf der Jahrestagung konnte der fruchtbare Erfahrungs- und Informationsaustausch unter den Anwendern fortgesetzt werden. In den Vorträgen wurde über neue Ergebnisse und Arbeiten der SAK-Mitglieder auf dem Programm- und Gerätesektor berichtet. Von der Firma Siemens wurde über neue und geplante Entwicklungen informiert. In der Diskussion zwischen Anwendern und Siemens wurden aktuelle Probleme und Fragen erörtert. Die Vorträge der Tagung sind in diesem Bericht zusammengefaßt.

Dem Kernforschungszentrum Karlsruhe sowie dem Hause Siemens sei im Namen des Anwenderkreises für die großzügige Unterstützung herzlich gedankt, die die Durchführung der Tagung und die Herausgabe des Tagungsberichtes ermöglichte.

Die 10. Jahrestagung wird 1979 beim Hahn-Meitner-Institut für Kernforschung, Berlin, stattfinden.

Udo Voges
Tagungsbeauftragter und Herausgeber

JAHRESTAGUNG SIEMENS-PROZESSRECHNER-ANWENDERKREIS

SAK 78

PROGRAMM

Mittwoch, den 5. April 1978

14.00 Uhr	Prof. F. Pieper, SAK-Vorstand E r ö f f n u n g	7
	Prof. Dr. W. Klose, KfK-Vorstand B e g r ü ß u n g	-
	U. Voges, SAK-Tagungsbeauftragter T a g u n g s i n f o r m a t i o n	-
14.30 Uhr	K. Heim, Universität Karlsruhe Ein Kommunikationsnetz zum Anschluß von Endperipherie an mehrere Rechnersysteme	8
15.00 Uhr	P. Schroth, Universität Karlsruhe Struktur der Ein-/Ausgabetreiber in einem universellen Kommunikationsrechner	19
15.30 Uhr	D. Eckert, Universität Karlsruhe Implementierungsverfahren für Emulatoren von Leitungsmultiplexern am Beispiel eines UNIVAC-Terminalmultiplexers (MUX-U100)	34
15.50 Uhr	K a f f e e p a u s e	
16.10 Uhr	E. Jabs, ESOC Darmstadt Rechnernetzwerke zur Satellitendatenverarbeitung unter Verwendung von S330 Rechnern Das GEOS/OTS Rechnersystem	-
16.40 Uhr	E. de Jong, ESOC Darmstadt Das METEOSAT Rechnersystem	-

16.50 Uhr	L. Gmeiner / W. Lemperle / U. Voges, KfK-IDT Karlsruhe Mehrfachimplementierung eines Reaktorschutz-programms in PHI1, PASCAL und IFTRAN auf der Siemens 330	48
17.00 Uhr	W. Geiger, KfK-IDT Karlsruhe Analyse, Test und Dokumentation von FORTRAN-Programmen mit dem automatischen Testsystem RXVP/S330	59
17.20 Uhr	H. Buseck, Bundesanstalt für Straßenwesen, Köln Einsatz einer S310 für Analogdatenverarbeitung	75
17.40 Uhr	H.-J. Schuster, PTB Braunschweig Ein Interface für den Blockverkehr über die E/A-Schnittstelle des Siemens-Prozeßrechners 330	90
18.00 Uhr	Schluß	

Donnerstag, den 6. April 1978

9.00 Uhr	G. Grüssinger / T. Wenzel, Universität Karlsruhe / Siemens Karlsruhe BASIC-Interpreter für S310 und R-Maschinen	99
9.30 Uhr	B. Uhlmann, HMI Berlin Erweiterte Dateiorganisation für den S310-BASIC-Interpreter	116
9.50 Uhr	L. Loup, KFA Jülich FLOARP, ein Floppy-Arbeitsprogramm für die Floppy-Disk an der Siemens 330	122
10.10 Uhr	K a f f e e p a u s e	
10.30 Uhr	G. Zöller, Siemens Karlsruhe Das Mikrocomputersystem 210	137

11.00 Uhr	J. Rinder, Siemens Karlsruhe Neue Komponenten der SIEMENS-Systeme 300-16 bit	149
11.30 Uhr	H. Schoknecht, Siemens Karlsruhe PEARL 300: Kompiliersystem und Verwendung für Echtzeitaufgaben	161
11.50 Uhr	K.-H. Drawert / K. Pieper, Siemens Karlsruhe Daten- und Funktionsverbund mit Rechnern des Siemens-Systems 300-16 bit	176
12.10 Uhr	Diskussion mit der Firma Siemens	
13.00 Uhr	M i t t a g p a u s e	
14.00 Uhr	SAK-Vollversammlung	
15.00 Uhr	Besichtigungen	
19.00 Uhr	Kaltes Büfett im Moninger "Malzboden", Ecke Kaiserstraße/Karlstraße, Begrüßung durch Direktor H. Klingler, Siemens K'he. u. Bürgermeister Prof. Dr. Seiler	

Freitag, den 7. April 1978

9.00 Uhr	G. Koch / R. Hoffmann, Universität Karlsruhe Statistische Angaben zum Stand der Prozeßdatenverarbeitung in der Bundesrepublik Deutschland	190
9.20 Uhr	K. Weise, PTB Braunschweig Rechnerkopplung Tektronix 4051 - Siemens 330	206
9.40 Uhr	P. Gais / K. Rodenacker / W. Abmayr / G. Schwarzkopf, GSF Neuherberg Mikroskopbildverarbeitung mit dem Prozeßrechner S330 und dem Programmsystem DIBIVE	215
10.00 Uhr	F.-J. Polster, KfK-IDT Karlsruhe FADABS: Ein Datenbanksystem für den Siemens Prozeßrechner 330	227

10.20 Uhr	R. Kerpe, KfK-IDT Karlsruhe Eine Implementierung des 'Least Recently Used'-Seitenersetzungsalgorithmus für den Prozeßrechner 330	245
10.30 Uhr	E. Bachner, KFA Jülich Anschluß eines graphischen H.P.-Terminals an eine S330	-
10.40 Uhr	K a f f e e p a u s e	
11.00 Uhr	U. Schwan, TH Aachen Mikrocomputer im Rechen- und Steuerwerk einer DVA301 zur Simulation einer DVA305	255
11.20 Uhr	W. Kristen, TH Aachen Kurven- und Prozeßsichtgerät für eine DVA301	265
11.40 Uhr	H. Huppertz / P. Reinhart / F. Rongen / W. Tenten, KFA Jülich Speichererweiterung für die Siemens 306	272
11.50 Uhr	F. Pieper, FHS Ulm Ein universeller Cross-Assembler für Mikrocomputer	280
12.00 Uhr	W. Becker, GH Paderborn Anschluß eines Trommelspeichers an die Arbeitsspeichernahtstelle der S301	297
12.10 Uhr	S c h l u ß w o r t	

Meine sehr geehrten Damen und Herren,

im Namen des SAK-Vorstandes begrüße ich Sie sehr herzlich und eröffne die neunte Jahrestagung des Siemens-Prozeßrechner-Anwenderkreises hier im Tagungsgebäude des Kernforschungs-Zentrums Karlsruhe.
Unseren Dank im voraus für die uns hier zuteil werdende Gastfreundschaft verbinde ich mit der Hoffnung, daß uns dieser gastliche Ort zum fruchtbaren Gedankenaustausch inspirieren möge.
Diese Hoffnung ist nicht ganz unbegründet - gibt es doch offensichtlich eine sehr enge Verwandtschaft zwischen Kernphysik und Informatik, die aus unerfindlichen Gründen bisher weithin übersehen wurde:
In beiden Disziplinen gibt es den zentralen Begriff des Wirkungsquerschnitts. Er ist ein Maß für die Bitdichte, d.h. für die Anzahl der auf ein Medium eintreffenden Anzahl bits einer Information, bezogen auf die Fläche, also die Aufnahme kapazität dieses Mediums.
Da ein bit fraglos die kleinste uns bekannte, ja sogar die kleinste denkbare Informationseinheit ist, kleiner noch als das kleinste Elementarteilchen, kann die Informatik für sich die Priorität beanspruchen, woraus sich zweifellos - die Kernphysiker mögen mir verzeihen - auch der fiktive Charakter des Wirkungsquerschnitts erklärt.
Andererseits müssen wir als Informatiker neidvoll anerkennen, daß erst die Kernphysiker diesen Begriff so recht mit Leben zu füllen imstande waren, woraus sich wiederum erklärt, warum wir Informatiker bislang noch keinen Resonanzquerschnitt haben vorweisen können, warum uns m.a.W. die Konvergenz der Theorie zur Praxis in weiter Ferne zu verschwimmen scheint.

Daher wiederum können wir an diesem Orte neue Hoffnung schöpfen, und in diesem Sinne fühle ich mich ermutigt, uns für den Verlauf der neunten SAK-Jahrestagung einen

angenehmen Wirkungsquerschnitt

zu wünschen!

Ein Kommunikationsnetz zum Anschluss von Endperipherie an mehrere Rechnersysteme

K. Heim
Universitaet Karlsruhe
Rechenzentrum

Kurzfassung

Der Anschluss von verteilter Endperipherie an Rechenanlagen erfordert ein entsprechendes Leitungssystem. Soll Endperipherie an mehrere Rechenanlagen gleichzeitig angeschlossen werden, so muss wahlweise eine Verbindung zwischen dem betreffenden Geraet und der gewuenschten Rechenanlage geschaltet werden. Waehrend im Fall der staendigen Verbindung zwischen der Endperipherie und der Rechenanlage immer die gleichen Partner, Geraet und Dialog- bzw. RJE-Eingang, miteinander verkehren, sind es beim dynamischen Verbindungsschalten wechselnde Partner. Ein Anschlusssystem fuer Endperipherie mit der Eigenschaft der Verbindungsschaltung wird im folgenden "Kommunikatiosnetz" genannt.

Die Aufgabe eines Kommunikationsnetzes ist somit, Teilnehmer miteinander zu verbinden, den Transport der Nachrichten zwischen ihnen durchzufuehren und ein Kommunikationsprotokoll abzuwickeln. Um den freizuegigen Verkehr unter allen Partnern zu ermoeglichen, wird ein einheitliches Kommunikationsprotokoll zwischen allen Partnern, Geraeten und Rechner-Eingaengen, das sog. virtuelle Terminal, eingefuehrt.

Die Verteilung von Endperipherie hat ein verzweigtes Leitungssystem zur Folge und benoetigt mehr Geraete fuer eine gleichwertige Versorgung der Benutzer gegenueber einer Zentralisierung der Endperipherie. Um diesem Mehraufwand an Hardware entgegen zu wirken, ermoeglicht das Konzept des nachfolgend beschriebenen Kommunikationsnetzes mittels Leitungs- und Geraetesharing systemoekonomische Konfigurierungen von Endperipherie. Das Anschlusssystem erlaubt Geraete aller Art mit anwendungsgerechter Leistung am Arbeitsplatz des Benutzers einzusetzen, wobei aus Gruenden der Wirtschaftlichkeit Geraete von mehreren Benutzern genutzt werden koennen.

Die Implementierung des Kommunikationsnetzes erfolgte auf einem Kleinrechnernetz Siemens 330/310 und wird im Gelaende der Universitaet Karlsruhe zum Anschluss von Endperipherie an die Grossrechenanlagen des Rechenzentrums (UNIVAC 1106/1108, Burroughs 7700) eingesetzt.

Inhalt
------

1. Einleitung
2. Zielvorstellung
3. Hardware
4. Logische Struktur
5. Virtuelles Terminal
6. Hoehere Protokolle
7. Kommunikationsnetzdienste

1. Einleitung

Unter Endperipherie von Rechnersystemen sind alle Geraete zu verstehen, ueber die der Benutzer seine Auftraege an die Rechenanlagen eingibt, und umgekehrt das Rechnersystem die Ergebnisse an den Benutzer ausgibt. Sie sind somit die Schnittstelle zwischen Benutzer und Rechenmaschine. Hierzu werden ueblicherweise folgende Greaete verwendet: Kartenleser/stanzer, Lochstreifenleser/stanzer, Drucker, Bildschirme, Schreibmschinen, Plotter, Digitalisierer usw.

Im Fall des Rechenzentrums der Universitaet Karlsruhe ist die gesamte Endperipherie nicht mit einer Rechenanlage zu verbinden sondern mit mehreren Rechenanlagen verschiedenen Typs. Hierzu ist ein entsprechendes Leitungssystem und ein Anwahlmechanismus erforderlich, der die Endgeraete per Befehl mit dem gewuenschten Rechner verbindet.

Ein Benutzer erwartet von einer Rechenanlage eine gewisse Leistung, d.h. er moechte die Ergebnisse seiner Auftraege an die Rechenmaschine innerhalb einer bestimmten Zeit erhalten, andernfalls ist die Benutzung der betreffenden Rechenmaschine fuer ihn unwirtschaftlich. Neben anderem beeinflust die Entfernung Arbeitsplatz - Rechenmaschine, genauer die Entfernung zur naechsten Ein/Ausgabe-Station die Zeit bis zum Erhalt der Antwort. D.h. die Endperipherie muss im Gelaende entsprechend den vorhandenen Bedarfsstellen verteilt werden, um eine flaechendeckende Versorgung zu erreichen.

Hierbei ist wichtig, dass der Benutzer alle fuer die Erledigung seiner Auftraege benoetigten Endgeraete in der Naehe seines Arbeitsplatzes zur Verfuegung hat. Im allgemeinen genuegt hier nicht, dass ein Bildschirm oder eine Schreibmaschine am Arbeitsplatz aufgestellt wird, es muessen alle erforderlichen Geraetefunktionen (Drucken, Plotten, Eingabe von Daten auf verschiedenen Datentraegern, usw.) bereit gestellt werden [HEIM78].

2. Zielvorstellung

Das zu konzipierende Endperipheriesystem soll die Dezentralisierung der Endgeraete in der Weise erlauben, dass anwendungsgerechte und geraeteoekonomische Konfigurierungen der Endperipherie an den Bedarfsstellen ermoeglicht wird. Die Verteilung von Endperipherie erfordert gegenueber einer zentralen Aufstellung zusaetzlich ein Leitungssystem und hat eine groessere Auffaecherung der Endperipherie in mehr Geraete zur Folge. Diesem Mehrbedarf an Hardware bei dezentraler Endperipherie kann teilweise durch Leitungs- und Geraetesharing begegnet werden.

Leitungssharing bedeutet gemeinsames Nutzen von Datenuebertragungsleitungen durch mehrere Datenstroeme, wobei die Uebertragungskapazitaet der Leitungen unter Beruecksichtigung der statistisch schwankenden Verkehrsintensitaeten der einzelnen Datenstoeme dimensioniert wird. Hierbei sind fuer eine gleichmaessigen Auslastung der Leitungen die Datenstroeme des RJE, Dateitransfer, gespoolte Druckausgabe usw. mit denen des Dialogsitzungen auf den Leitungen zu mischen.

Die Anforderungen der Benutzergemeinschaft eines Universitaet- rechenzentrums an den Zugang zu den Rechenanlagen erlauben eine Konzentrierung von Geraeten in mehreren sog. Terminalpools im Gelaende, was Voraussetzung fuer ein Geraetesharing ist. Hierzu werden mehrere Bildschirme in einem Raum zusammen mit den erforderlichen Zusatzgeraeten, wie Drucker, Plotter und Disketten, aufgestellt und eine gemeinschaftliche Nutzung der Zusatzgeraete durch Benutzer des betreffenden Pools ermoeglicht. Die Geraete sollen hierbei sowohl fuer Dialog als auch fuer Stapelein/ausgabe verwendbar sein.

Die Art der Schnittstellen der Rechnersysteme zur Eingabe von Auftraegen und Rueckgabe der Ergebnisse hat Einfluss auf die Art der Endperipherie. Rechnersysteme ermoglichen fuer den Benutzer grundsaetzlich zwei Arten von Schnittstellen: Dialog- und Stapelein/ausgabestation.

Eine Dialogstation besteht aus einer Tastatur, Druckwerk oder Bildschirm und bedarfsweise aus weiteren Zusatzgeraeten. Eine Stapelein/ausgabestation ist geeignet fuer die Eingabe von Auftraegen, die auf Datentraegern vorbereitet sind und bei denen nicht eine unmittelbare Antwort von der Rechenanlage erwartet wird. Sie besteht aus Eingabegeraet, Drucker und Konsole.

Beide Arten von Schnittstellen sollen ueber das Endperipheriesystem fuer den Benutzer zugaenglich sein, wobei die Geraete wahlweise als Zusatzgeraete fuer die Bildschirme oder Ein/Ausgabe-Geraete fuer den RJE verwendet werden sollen.

3. Hardware

Zur Realisierung der vorgegebenen Ziele fuer den Entwurf des Kommunikationsnetzes ist zunaechst eine hierfuer geeignete Hardware zu waehlen. Um die teilweise sehr komplexen Funktionen in sinnvoller Weise implementieren zu koennen, wurde ein Kleinrechnernetz als Leitungs- und Peripherie-Anschlussystem gewaehlt (Bild 1).

Hierbei ist vorgesehen, fuer jeden Terminalpool einen Kleinrechner in der Funktion eines Leitungskonzentrators und einer Anpassungssteuerung fuer die Geraete einzusetzen, sog. Stationsrechner. Er wird nach den Beduerfnissen seiner Benutzer mit Bildschirmen, Druckern, Plottern und Disketten ausgeruestet. Darueberhinaus sind Anschluesse fuer weitere Geraetetypen ohne Schwierigkeiten moeglich.

Die Aufgabe des Stationsrechners ist es, das Leitungssharing (Multiplexer) fuer den Datenverkehr mit den Grossrechenanlagen und das Geraetesharing durch das wahlweise Zuschalten als Zusatzgeraeten zu den Bildschirmen und durch ihre alternative Verwendung als RJE-Geraete durchzufuehren.

Vor den Grossrechenanlagen ist ein weiterer Kleinrechner installiert, der die fuer den Verkehr mit den Grossrechenanlagen notwendigen Leitungsprozeduren emuliert und den Verkehr mit den Terminalpools abwickelt, der sog. Vorschaltrechner.

Die Stationrechner sind mit dem Vorschaltrechner ueber durchgeschaltete Telefonleitungen verbunden. Die Verwendung von Telefonleitungen eruebrigt das kostspielige Verlegen von eigenen Leitungen und garantiert eine hohe Mobilitaet des Endperipheriesystems.

4. Logische Struktur

Durch die Vielzahl von Komponenten und den erforderlichen hohen Parallelausfuehrungsgrad eines Kommunikationssystems kommt der Software-Architektur eines solchen Systems eine besondere Aufmerksamkeit zu. Um die Komplexitaet des Systems klein zu halten, werden so wenig wie moeglich verschiedene Schnittstellen zwischen den Komponenten verwendet. Der konstruktive Aufbau des Systems aus Funktionseinheiten ist hierbei in sofern wichtig, als dadurch die Moeglichkeiten der Konfigurierungen verschiedener Systeme, die Austauschbarkeit von Funktionseinheiten und Erweiterbarkeit des Systems durch neue Funktionseinheiten vorgegeben werden. Durch die physikalische Trennung der Systemkomponenten, Vorschalt- und Stationsrechner, wobei in allen Teilen funktionsgleiche Module vorhanden sind, ist die Konfigurierung ein wichtiges Mittel zur System-Erstellung.

Die Funktionen, die vom System auszufuehren sind, lassen sich folgendermassen gliedern:

1. Ein/Ausgabe

- Verkehr mit Grossrechenanlagen nach gegebenen Leitungs-Protokollen
- Datenuebertragung zwischen Stations- und Vorschaltrechnern ueber serielle Leitungen
- Datenverkehr mit Geraeten

2. Verbindungsschaltung

   zwischen beliebigen Dialog- und RJE-Stationen der Rechenanlagen und Endgeraeten

3. Hoehere Kommunikationsprotokolle

   fuer RJE, Druckausgabe und Dateitransfer jeweils fuer jede der angeschlossenen Grossrechenanlagen

4. Netzdienste

- Zugangskontrollsystem fuer die Benutzer der Endperipherie
- Editieren von Disketten

Aufgrund der vorhandenen Vielfalt von Geraetetypen und Leitungsprotokollen und der Erweiterbarkeit des Systems durch weitere Geraetetypen und Rechneranschluesse ist es sinnvoll, zur Vereinfachung des Verfahrens der Verbindungsschaltung und zur Implementierung der hoeheren Kommunikations- protokollen und Netzdienste, eine Transformation aller Ein/Ausgabeoperationen in eine einheitliche und datenstromspezifische Schnittstelle durchzufuehren. Diese Schnittstelle trennt damit die Ein/Ausgabeprozesse von den uebrigen Prozessen des Systems. Sie wird als Port des Kommunikatinsnetzes bezeichnet (Bild 2).

Port

ist die Einheitsschnittstelle des Kommunikationsnetzes, ueber die der Verkehr mit allen Geraeten und ueber die Leitungsschnittstellen mit den Grossrechnern und den Stationsrechnern der Terminalpools abgewickelt wird. Die Schnittstelle ist voll duplex, d.h. die Eingabeoperationen werden unabhaengig von den Ausgabeoperationen ausgefuehrt. Die Anpassung der jeweiligen Ein/Ausgabeoperationen des betreffenden Geraets bzw. Leitungsschnittstelle an die Portschnittstelle nimmt der sog. Porttreiber vor. Dabei sind unter Umstaenden Leitungsprotokolle, Multiplexerfunktionen bzw. Dateizugriffsverfahren zu implementieren.

Task

Die auf den Port aufgesetzten Prozesse zur Ausfuehrung von Netzfunktionen, wie Verbindungsschaltung, Datentransfer, RJE-Protokolle, Druckausgabe auf die Stationsdrucker, Diskketteneditor usw., werden Task genannt. Aufgrund der Einheitsschnittstelle 'Port' und eines dynamischen Zuweisungsmechanismus von Ports zu logischen Nummern, die von den Tasks bei den Portoperationen benutzt werden, sind die Tasks unabhaengig von der jeweiligen Art des Ein/Ausgabegeraets bzw. Grossrechners. Mit der Hilfe eines fuer alle Tasks anwendbaren Mechanismus zum Steuern dieser (Starten und Anhalten) und des Zuweisungsmechanismus von Ports ist das Schalten von Verbindungen, das Umsteuern von Datenstroemen und die Festlegung einer Uebrwachungskonsole fuer jede Task zur Meldung von Geraetedefekten realisiert.

5. Virtuelles Terminal

Bildschirme haben die Eigenschaft, dass sie neben der Ausgabe von Zeichen verschiedene Steuerfunktionen ausfuehren koennen, z.B. Steuerung der Bildschirmmarke, Einstellung der Darstellung von Zeichen, Loeschen des Bildschirms usw. Bedauerlicherweise bestehen hierbei Unterschiede zwischen Bildschirmen verschiedenen Typs sowohl in den Funktionen als auch in ihrer Codierung.

Die Regeln zum Austausch von Daten und Steuerinformationen bezeichnet man auch als Kommunikationsprotokoll zwischen Bildschirm und Anwenderprozess [BARB77, SCHIK76]. Hierdurch entsteht eine Abhaengigkeit zwischen den beiden miteinander kommunizierenden Partnern. Fuer einen freizuegigen Verkehr zwischen beliebigen Partnern muss eine einheitliche Festlegung eines solchen Protokolls erfolgen. Zu dieser Festlegung wird ein hypothetisches Terminal entworfen, das sog. virtuelle Terminal. Da die Funktionen des virtuellen Terminals nicht in der gleichen Art bei allen Terminals vorhanden sind, muessen sie auf den einzelnen Geraeten durch die Geraetetreiber emuliert werden.

Das virtuelle Terminal wird im wesentlichen festgelegt durch

- das Format des Bildschirms (Anzahl, Zeichen/Zeile, Anzahl Zeilen)
- Adressierung von Bildschirmbereichen wie Zeilen und Ausschnitte von Zeilen
- Codierung der Zeichen und der Bildschirmbefehle fuer die Bewegung der Bildschirmmarke, das Einschalten des Schreibschutzes und der verschiedenen Zeichendarstellung, Bildschirm und Bildschirmbereiche loeschen, usw.

6. Hoehere Protokolle

Unter hoeheren Protokollen versteht man Kommunikationsprotokolle, die auf Leitungsprotokollen aufbauend zwischen zwei Partnern, im allgemeinen Prozesse, abgewickelt werden. Anwendungsfaelle solcher Protokolle sind RJE-Protokolle, Dateitransferprotokolle usw.

Im Kommunikationsnetz sind die Tasks fuer die Realisierung von hoeheren Protokollen eingerichtet worden. Da die Port-Schnittstelle diese Prozesse unabhaengig von den Leitungsprotokollen bzw. von dem geraetespezifischen Ein/Ausgabeverkehr macht, koennen diese Protokolle daher fuer das gesamte Kommunikationsnetz einheitlich realisiert werden.

Auf diese Weise wurden die Protokolle fuer den RJE, fuer das Ausgeben der Druckausgabe der Batch- Programme der Grossrechnern auf den Stationsdrucker und ein Dateitransferprotokoll zwischen den Disketten und Grossrechenanlagen implementiert.

7. Kommunikationsnetzdienste

Darunter sind alle Dienste zu verstehen, die das Kommunikationsnetz unabhaengig von den angeschlossenen Grossrechnern erbringt:

Verbindungsschaltung

Per Befehl am Terminal kann eine Verbindung zwischen dem betreffenden Geraet und einem hierbei angewaehlten Grossrechner geschaltet werden. Der Mechanismus ist allerdings so allgemein, dass er das Schalten von Verbindungen zwischen beliebigen Teilnehmern des Kommunikationsnetzes erlaubt, d.h. von Geraet zu Grossrechnereingang, von Grossrechnereingang zu Grossrechnereingang und von Geraet zu Geraet.

Es werden hierbei entsprechend der Anforderung sog. Einweg- oder Zweiwegverbindungen geschaltet, bei denen nur in eine oder in beide Richtungen Datentransfer stattfindet. Fuer jede Verbindung wird eine Ueberwachungskonsole hierbei festgelegt, ueber die Kommunikationsnetzmeldungen ausgegeben werden, wie Geraetedefekte und Leitungsausfaelle. Diese Ueberwachungskonsole kann ein beliebiges Geraet sein, welches auch im Betrieb nach Belieben umdefiniert werden kann. So ist es moeglich, beispielsweise die Ueberwachung aller RJE-Geraete und Drucker ueber einen Bildschirm durchzufuehren.

Benutzung von Zusatzgeraeten

Das Geraetesharing erfolgt durch Anforderung von Zusatzgeraeten des betr. Terminalpools durch den Benutzer. Hierbei werden die jeweiligen Datenstroeme von einem Geraet zum anderen umgeschaltet oder ein paralleler Datenstrom auf ein zusaetzliches Geraet geschaltet. Auf diese Weise kann der Benutzer die Bildschirmausgabe auf den Drucker oder auf eine Datei der Diskette und die Eingabe des Bildschirms auf die Eingabe von Lochstreifenleser oder Datei der Diskette umsteuern.

Anhalten und Fortsetzen von RJE-Prozessen und Druckausgabe

Das gemeinsame Nutzen von Geraeten als Zusatzgeraete fuer Dialogbildschirme als auch fuer RJE und Druckausgabe zu und von den Grossrechnern erfordert die Moeglichkeit der Unterbrechung von langdauernden Druckausgaben bzw. Eingaben von Batch-Programmen. Hierfuer ist der Mechanismus des Anhaltens und Fortsetzens der RJE-Prozesse vorgesehen. Der Benutzer am Bildschirm kann auf diese Weise die Druckausgabe eines Batch-Programms unterbrechen, um den Drucker fuer die Erstellung einer Hardcopy seiner Bildschirmausgabe im Dialog zu verwenden und danach die Druckausgabe wieder fortsetzen.

Diskketteneditor

Die Diskettengeraete der Terminal-pools sind vorgesehen sowohl fuer die Dialogstationen als Zusatzgeraete als auch fuer die Stapeleingabe. Das letztere erfordert das Beschreiben und Korrigieren von Disketten-Dateien, die sinnvollerweise unabhaengig von den Grossrechenanlagen moeglich sein sollte. Hierzu wurde ein Diskketteneditor implementiert, der in den Stationsrechner ablaeuft. Hiermit koennen zeilen- und zeichenorientierte Textaufbereitung auf Disketten-Dateien vorgenommen werden.

Literatur

[BARB 77] Barker, D. L. A. :
The Role and Nature of Virtual Terminal
Computer Communication Review
July 1977, Vol. 7, No.3
pp. 59-76

[HEIM 77a] Heim, K.:
Versorgungsstruktur von Rechenleistung,
Festschrift zum 10-jaehrigen Bestehen des
Rechenzentrums der Universitaet Karlsruhe, 1977
pp. 26-33

[HEIM 77b] Heim, K.:
Dezentraler Zugriff fuer Dialog- und
Stapelverarbeitung eines Universitaetsrechenzentrums
mit Hilfe eines Kleinrechnernetzes,
GI-Workshop ueber Organisation von Rechenzentren,
1977, (erscheint im Tagungsband, Springer Verlag)

[HEIM 78] Heim, K.:
Planungsueberlegungen zum Endperipheriesystem von
Rechenanlagen,
Rechenzentrum Heft 1/78, 1978
Carl Hanser Verlag, Muenchen
pp. 14-17

[SCHIK 76] Schicker, P.; Duenki, A.:
Virtual Terminal Definition and Protocol,
Computer Communication Review, 1976
pp. 1-18

[SCHIK 77] Schicker, P.; Zimmermann, H.:
Proposal for a Scroll Mode Virtual Terminal,
Computer Communication Review,
July 1977, Vol. 7, No. 3
pp. 23-58

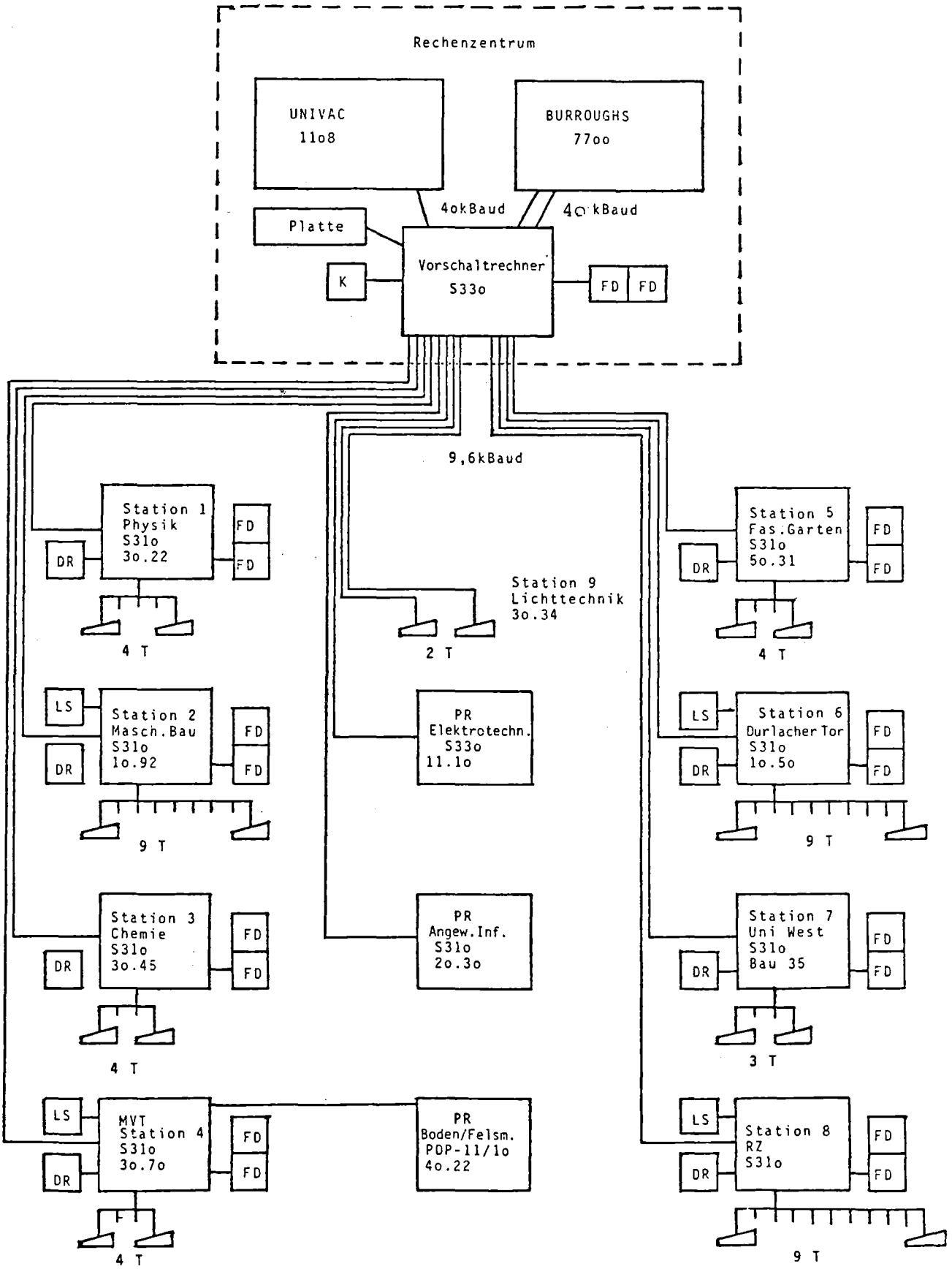
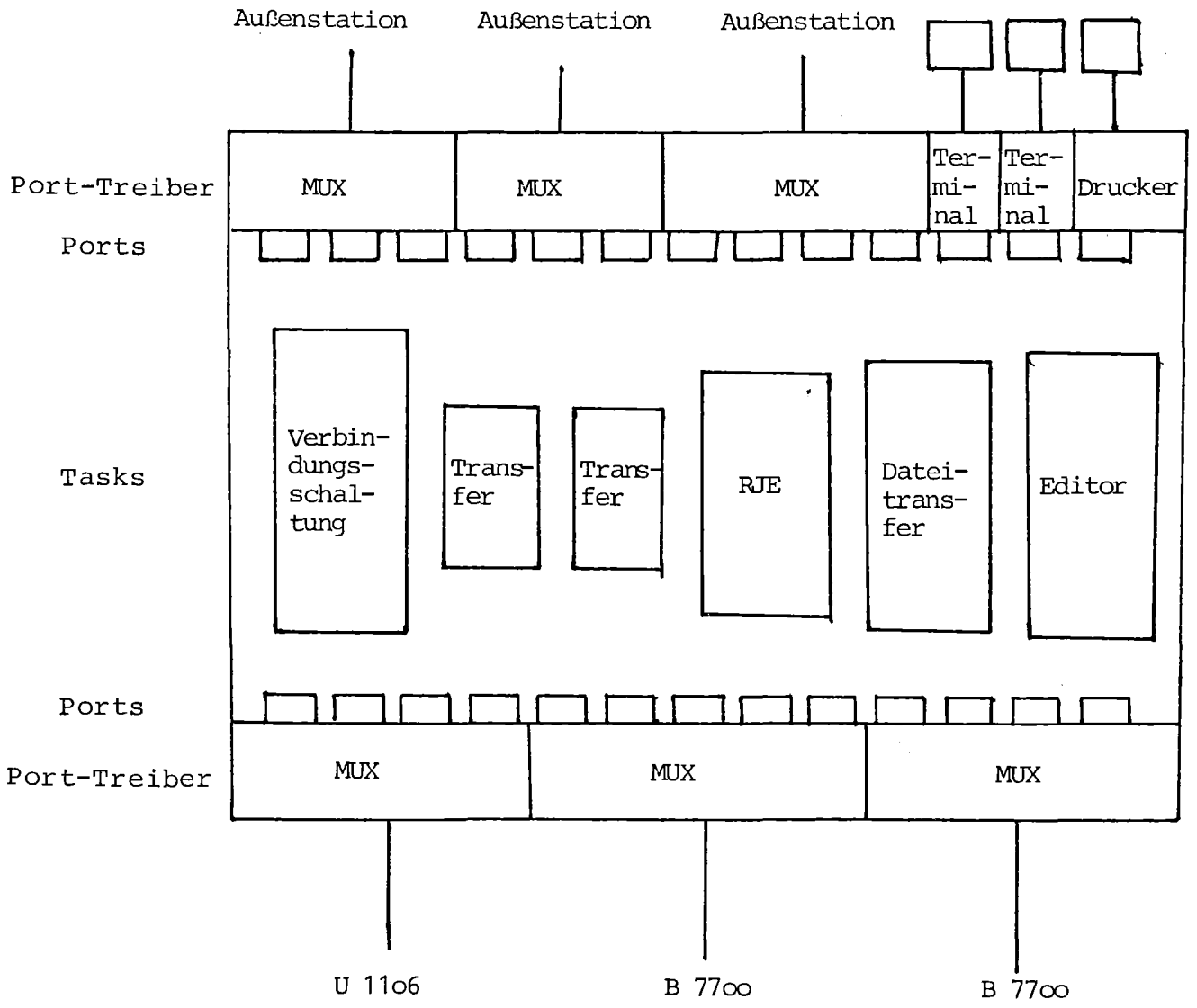


Bild 1: Kommunikationsnetz



MUX = Leitungsmultiplexer
RJE = Remote-Job-Entry
U 1106 = UNIVAC 1106
B 7700 = Burroughs 7700

Bild 2: Logische Struktur: Porttreiber und Tasks

Jahrestagung
Siemens-Prozessrechner-
Anwenderkreis
(vom 5.-7.April 1978
im Kernforschungszentrum Karlsruhe)

SAK 78

Manuskript zum Vortrag

Struktur der Ein-Ausgabetreiber in einem universellen Kommunikationsrechner

von

Dipl.Ing. Peter Schroth
Rechenzentrum
Universität Karlsruhe

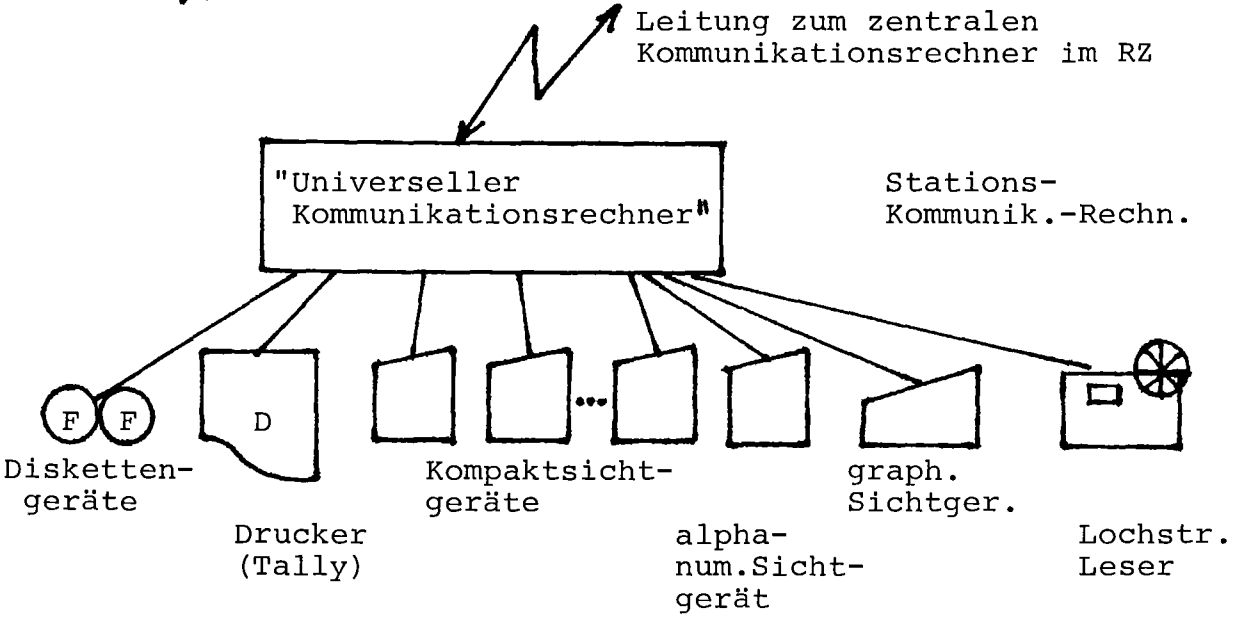
1. Einleitung

1.1 Themenstellung

E/A-Treiber sind diejenigen Baustein in einem System, welche die E/A-Aufrufe einer höheren Ebene (z.B. Benutzerebene) ausführen in der Maschinenebene (E/A-Maschinenbefehle, Unterbrechungsmechanismus der Maschine).
Werden diese Treiber einheitlich und streng modular aufgebaut, dann ergeben sich daraus erhebliche Vorteile, u.a. : leichte Erstellbarkeit neuer Treiber (z.B. Anschluß von Fremdgeräten und Fremdrechnern), große Änderungsfreundlichkeit, optimale Bedienung der angeschlossenen E/A-Geräte bzw. Leitungen.

1.2 Rahmen für das Thema

Entwicklung eines Kommunikationsnetzes an der Universität Karlsruhe.
"Universeller Kommunikationsrechner" (im Gelände der Universität Karlsruhe):



1.3 Forderungen an einen universellen Kommunikationsrechner

Hard- und Software muß so konzipiert sein, daß

1) ohne großen Aufwand angeschlossen und optimal bedient werden können :
   - alle Arten von Peripheriegeräten, insbesondere auch Fremdgeräte, (z.B. s. Bild "Universeller Kommunikationsrechner")
   - beliebige andere Rechner mit vorgegebenen Übertragungsprozeduren;
2) die gängigsten Datenübertragungseinrichtungen Verwendung finden können;
3) neben den rechner-spezifischen Schnittstellen wenigstens folgende Schnittstellen zur Verfügung stehen und softwaremäßig unterstützt werden :
   - V 24
   - Linienstrom 20 oder 60 mA,
     (2- und 4-Draht, mit oder ohne interne Quelle)
4) asynchrone und synchrone Gleichlaufverfahren möglich sind;
5) Voll- und Halbduplex-Betrieb möglich ist;
6) auch höhere Übertragungsraten (bis 50 K Baud) möglich sind;
7) auch eine größere Anzahl langsamer Geräte bedient werden kann (z.B. über Multiplexer);
8) Sonderzustände an den E/A-Schnittstellen erkannt und von der Software behandelt werden, z.B.
   - fehlende Schnittstellen-Signale bei V 24, beispielsweise
     - Sendebereitschaft (Clear to Send)
     - Betriebsbereitschaft (Data Set Ready)
     - Empfangssignalpegel (Data Carrier)

- Geräte - Sonderzustände
  bei Drucker beispielsweise Papierende
  bei LS beispielsweise Streifenende
  usw.
- Bit-Framing Fehler (z.B. Stopbit)
- Übertragungsfehler (Parity, BCC, CRC)
-"Timeouts"
-"Overruns"
-"BREAK"

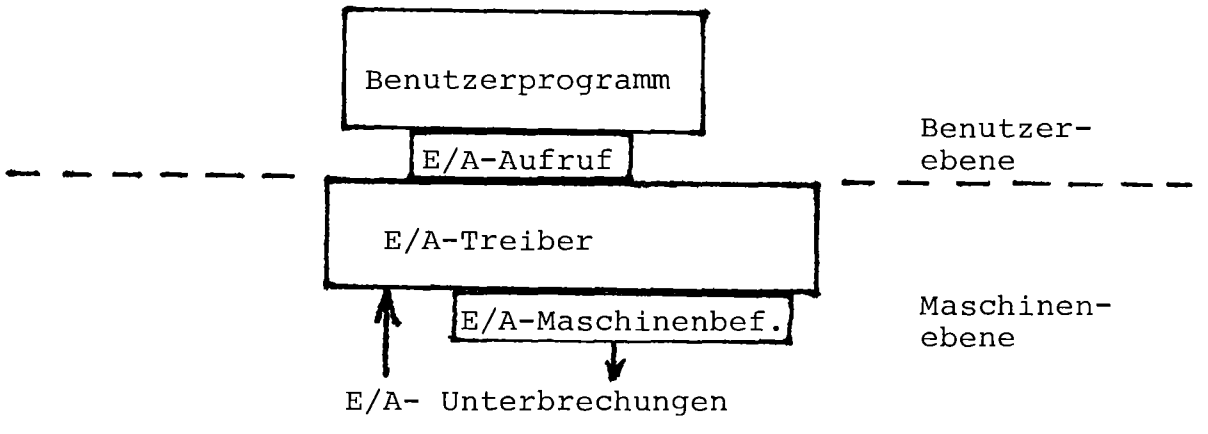
9) Umkonfigurierung von Geräten und Koppelstrecken ohne großen Aufwand möglich ist (eine Neugenerierung des Systems ist zu viel Aufwand !)

10) innerhalb des Systems ein "virtuelles Terminal" definiert ist. An der Schnittstelle zwischen Benutzerprogramm und E/A-Treiber sollen alle Daten im Format dieses "virtuellen Terminals" vorliegen. Die E/A-Treiber müssen für die Formatwandlung sorgen, falls erforderlich.

2. E/A-Treiber

2.1 Definition

E/A-Treiber sind diejenigen Bausteine in einem System, welche die E/A-Aufrufe einer höheren Ebene ausführen in der Maschinenebene

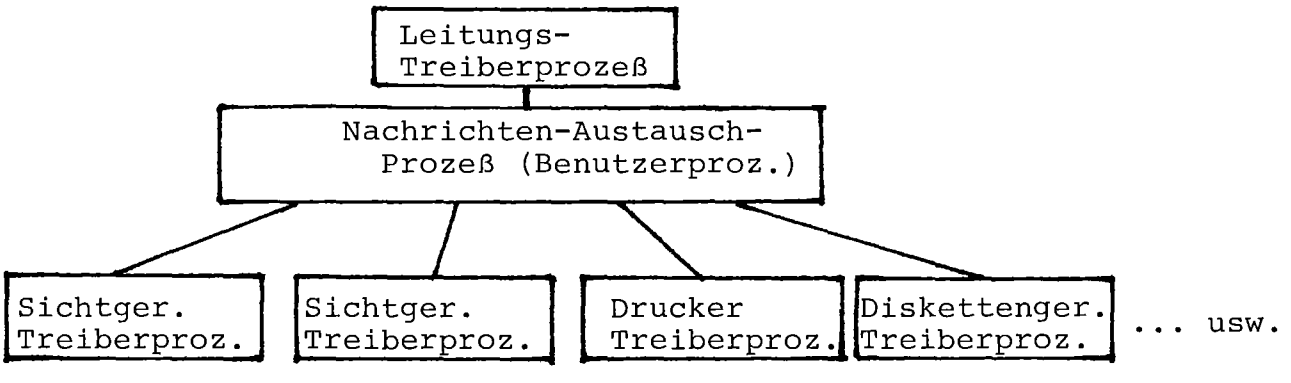


2.2 Forderungen an die E/A-Treiber

- Einheitliche Struktur aller E/A-Treiber
- Strenge Modularität, d.h. alle Treiber benutzen einen Satz von Basisbausteinen.
- Programmcode "eintrittsinvariant" (re-entrant)
  d.h. Programmcode für mehrere Peripheriegeräte gleichen Typs nur einmal vorhanden;
  Trennung zwischen Programmcode und Datenbereichen.
- Die komplette Unterbrechungsbehandlung soll in den Basisbausteinen "verschwinden".
  Der Treiber sieht dann nach außen hin so aus, als ob er nur ein einziges Gerät bedienen würde.
  Daraus folgt :
  - Hohe Änderungsfreundlichkeit
  - Leichte Erstellbarkeit neuer Treiber
  - Wohldefinierte Schnittstellen zur "höheren Ebene" (Benutzerebene).

2.3 Stellung der E/A-Treiber im System

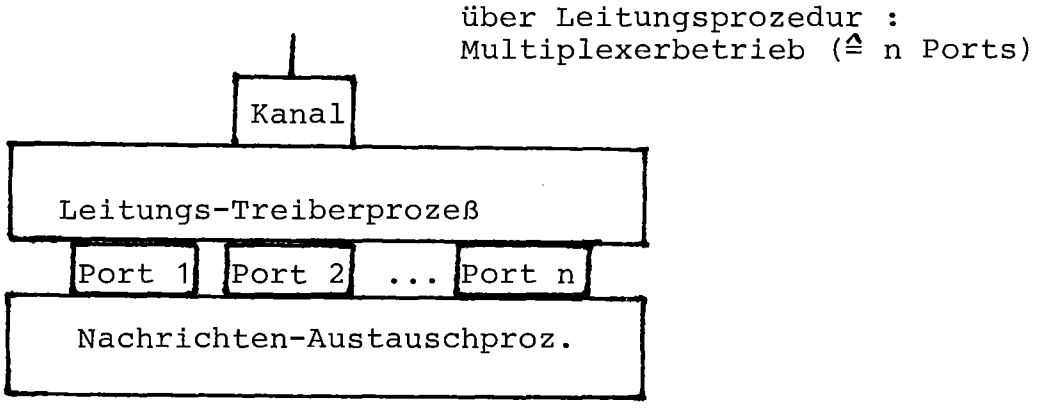
Die Treiber laufen in Form von Treiberprozessen im System ab.



- Alle Prozesse arbeiten asynchron zueinander.
- Der Datenaustausch zwischen den Prozessen erfolgt über Warteschlangen in Form von Nachrichten.

- Diese Nachrichten haben ein einheitliches Format. Hier ist auch der Code (USASCII) sowie die Bedeutung und Interpretation der Steuerzeichen festgelegt ("virtuelles Terminal"). Der Treiber führt die Formatwandlung durch.
- Jeder Treiberprozeß bedient genau eine E/A-Anschlußstelle (Kanal).
- Jeder Treiber kann jedoch einen oder mehrere Ports zum Nachrichten-Austausch-Prozeß hin bedienen. Das hängt davon ab, ob an der betr. E/A-Anschlußstelle ein oder mehrere Log.Geräte hängen.

Beispiel :

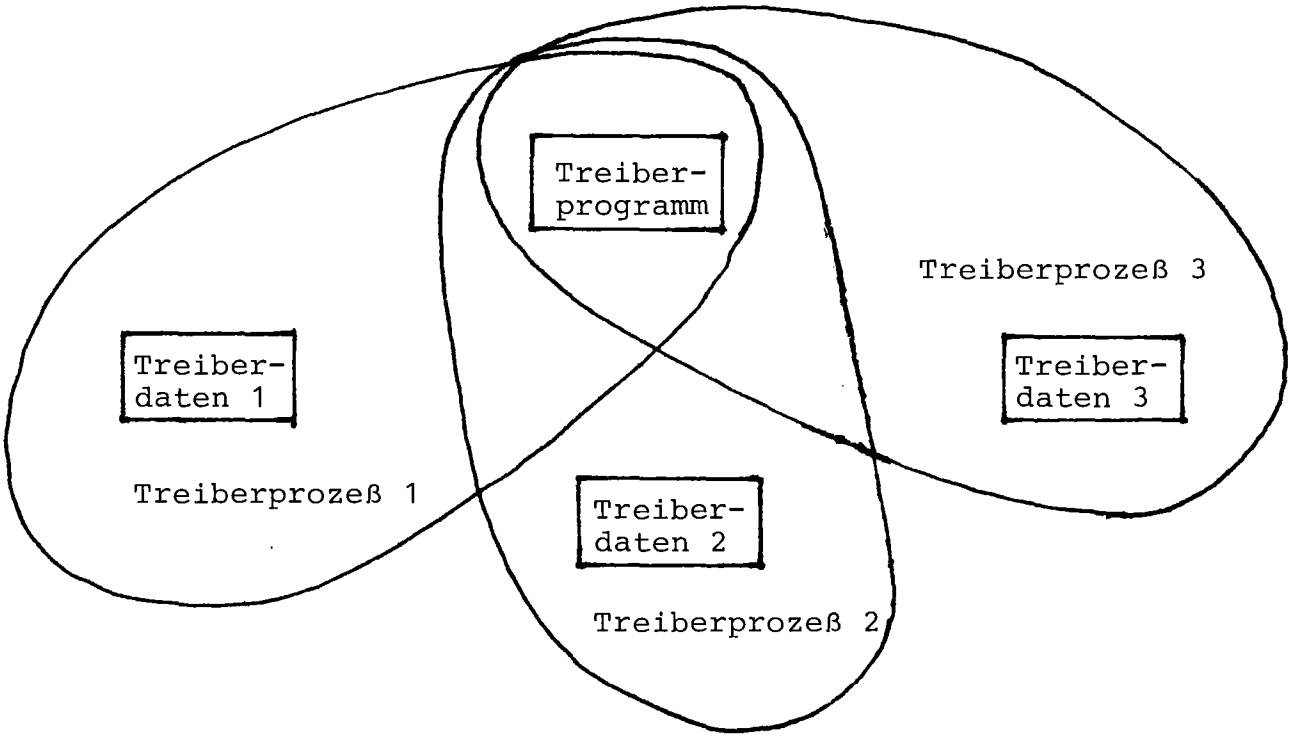


3. Treiberprozeß - Struktur

3.1 Grobstruktur

- Programmteil (invariant)
- Datenteil (variant)

Hieraus folgt :
für mehrere gleichartige Treiberprozesse muß der Programmteil nur einmal vorhanden sein.
Dagegen benötigt jeder Treiberprozeß seinen eigenen Datenteil.



Programmteil

Kann unterteilt werden in

- Treiber-Kontrolle
- Ausgabe-Request
- Eingabe-Request

Datenteil

Kann unterteilt werden in

- Kanaltafel
- Port-Tafel(n)
- Nachrichten-Puffer

3.2 Programmstruktur

3 Programmteile

1) Treiber-Kontrolle
   - hier befindet sich der Treiberprozeß im Ruhezustand;
   - von hier aus werden, getrieben durch Ereignisse, alle Treiberprozeß-Aktivitäten gestartet;
   - hier enden alle Treiberprozeß-Aktivitäten;
   - hier wird, falls mehrere Ereignisse gleichzeitig vorliegen entschieden, welches Ereignis zunächst bearbeitet wird;
   - hier wird entschieden, welcher Port als nächster an der Reihe ist.

   Folgende Ereignisse können einen ruhenden Treiberprozeß wekken :
   - Ausgabe-Request :
     Weckender Prozeß ist der Nachrichten-Austausch-Prozeß, wenn von diesem wenigstens eine Nachricht an den Treiber-Prozeß übergeben werden soll zum Zwecke der Ausgabe.
   - Eingabe-Request :
     Weckender Prozeß ist der Nachrichten-Austausch-Prozeß, wenn er den Treiber-Prozeß dazu auffordern will, von einem Gerät her eine Nachricht einzulesen.
     Nur interessant für passive Geräte !!!
   - Anruf :
     Urheber dieses Ereignisses ist der Kanal, wenn am Gerät eine Eingabe erfolgt (z.B. wenn eine Anruftaste gedrückt wird, wenn das erste Zeichen einer Nachricht eingetippt wird o.ä.)

Nur interessant für aktive Geräte, z.B. Sichtgeräte, die sich per Interrupt selbst melden können.

- Kanalfehler :
  Urheber dieses Ereignisses ist der Kanal (ähnlich wie beim Anruf), wenn irgend welche Fehler am Gerät, an der Koppelstrecke usw. erkannt werden.
  (Beispiel : das Verbindungskabel zu einem Rechner wird gezogen)

2) Ausgabe-Request
   In diesen Programmteil des Treibers wird verzweigt, wenn die Treiber-Kontrolle erkannt und entschieden hat, daß das Ereignis "Ausgabe-Request" behandelt werden soll.
   Aufgabe sind u.a.
   -Entgegennahme der auszugebenden Nachricht über den betreffenden Port vom Nachrichten-Austausch-Prozeß,
   -Umwandlung der Nachricht in eine ausgabegerechte Form entsprechend den Forderungen der E/A-Anschlußstelle (Kanal)
   -Evtl. Einpacken der Nachricht in ein Transportprotokoll entspr. der vereinbarten oder festgelegten Übertragungsprozedur
   -Evtl.Formatwandlung (virt.Terminalformat in gerätespezifisches Format)
   -Evtl. Umcodierung
   -Anstoß des eigentlichen Ausgabevorganges

3) Eingabe-Request
   In diesen Programmteil des Treibers wird verzweigt, wenn die Treiber-Kontrolle erkannt und entschieden hat, daß das Ereignis "Eingabe-Request" bei passiven Geräten bzw. das Ereignis "Anruf" bei aktiven Geräten behandelt werden soll.
   Aufgaben sind u.a.
   -Anstoß des eigentlichen Eingabevorganges
   -Evtl. Umcodierung
   -Evtl.Formatwandlung (gerätespezifisches Format in virt. Terminalformat)

-Evtl.Entpacken der Nachricht aus einem Transportprotokoll bei gegebener Übertragungsprozedur.
-Umwandlung der Nachricht von der Form, wie sie von der E/A-Anschlußstelle geliefert wird, in das einheitliche Format an der Schnittstelle zum Nachrichten-Austausch-Prozeß
-Übergabe der Nachricht über den betr. Port an den Nachrichten-austausch-Prozeß

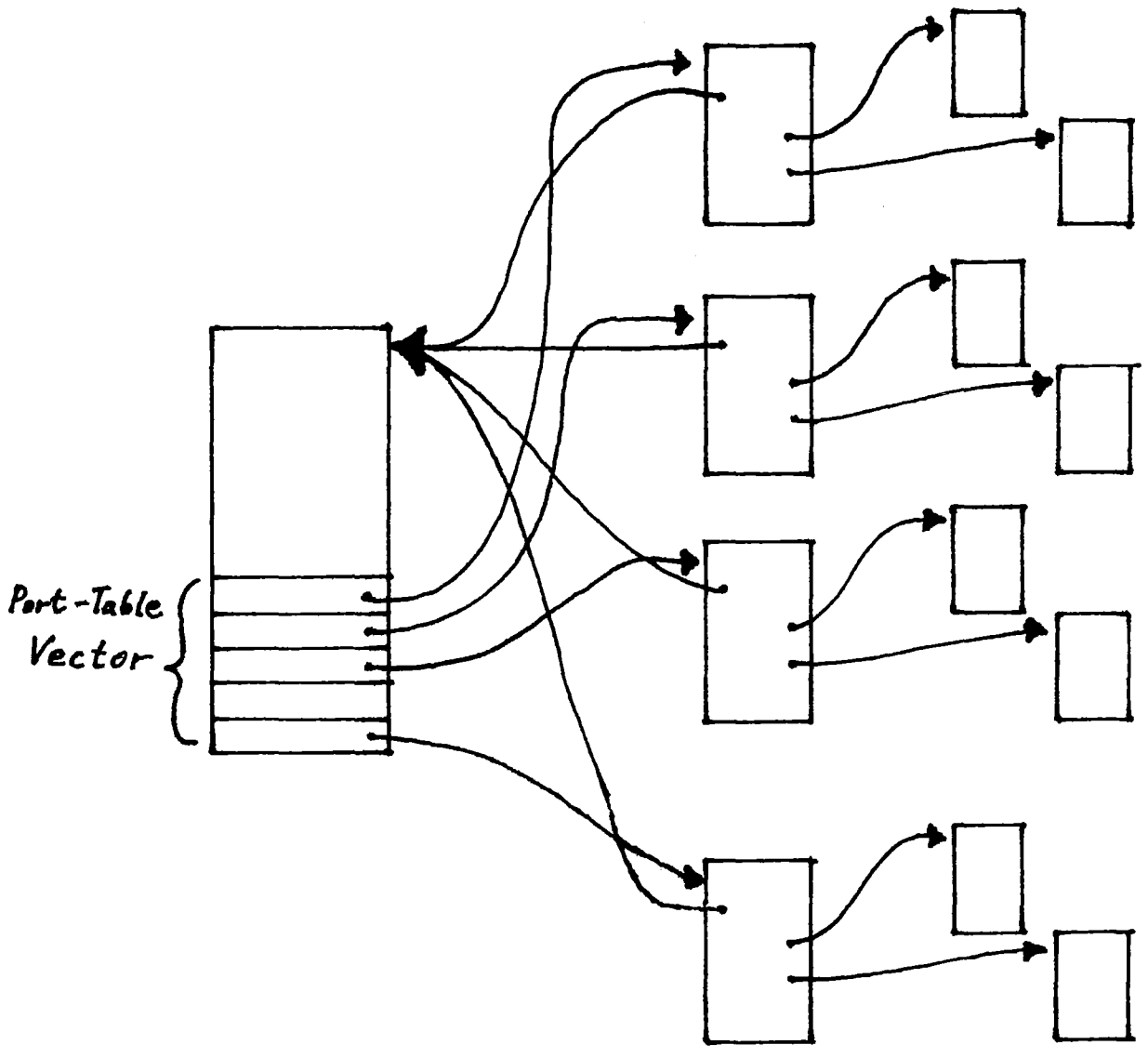
Anmerkungen

Die einzelnen Aufgaben lassen sich nicht immer eindeutig dem einen oder anderen Programmteil zuordnen. (So kann beispielsweise das Entpacken einer Nachricht aus einem Transportprotokoll schon in der Treiber-Kontrolle notwendig sein).
Verteilt über die einzelnen Programmteile sind beispielsweise Aufgaben wie
-Koordinierung der Übertragungsrichtungen bei Halb-Duplex-Betrieb
-Fehlerprüfung und Fehlerbehandlung
-Berücksichtigung von Verzögerungszeiten.

3.3 Datenstruktur

Primäre Datenbasis des Treiberprozesses ist die Kanaltafel CT (Channel Table). Sie enthält Zeiger auf die Port-Tafeln PT(Port Tables), wobei für jeden Port eine PT vorhanden ist. Jede Port-Tafel enthält Zeiger auf die Nachrichten-Puffer.

Datenstruktur des Treiber-Prozesses und Verzeigerung



Kanaltafel

Sie enthält folgende Daten :

-Adresse des Treiber-Kontroll-Programmes
-Zeiger auf die aktuell in Bearbeitung befindliche PT
-Kanal-Status-Information
-Zeiger auf den aktuell in Bearbeitung befindlichen Nachrichten-puffer (restlicher Puffer)
-Restliche Datenlänge im Nachr.-Puffer
-Bytemarkierung im Nachr.-Puffer
-Kanal-Anzeigen
-Nr. des Treiber-Prozesses
-Port-Tafel-Vektor
-Sämtliche Hilfsvariable für den Treiberprozeß

Port-Tafeln

Sie enthalten jeweils folgende Daten :

-Zeiger auf die Kanaltafel
-Status-Information über den Port
-Fehleranzeigen
-Nr. des Nachrichten-Austausch-Prozesses
-Zeiger auf den Nachrichten-Puffer für Eingabe
-Zeiger auf den Nachrichten-Puffer für Ausgabe
-sämtliche Port-spezifischen Hilfsvariablen für den Treiber-prozeß

Nachrichten-Puffer

Sie enthalten :

-den Text
-einen Nachrichten-Kopf (enthält z.B. die Länge des Textes)

4. Treiber-Bausteine

Liegen vor in Form von Unterprogrammen. Die wichtigsten Bausteine sind :

-IDLE
Der Treiber geht in den Ruhezustand und wartet auf das nächste Ereignis. Verzweigung entspr. des Ereignisses

-INITTRANSMIT
Initialisierung der (zeichenweisen) Übertragung zum Gerät bzw. zur Leitung.
Seitenausgang bei Fehlersituationen.

-TRANSMIT
Übertagung eines Zeichens.Seitenausg. bei Fehlern

-FINTRANSMIT
Abschluß der Übertragung

-INITRECEIVE
Initialisierung des (zeichenweisen) Empfanges vom Gerät oder der Leitung.
Seitenausgang bei Fehlersituationen.

-RECEIVE
Empfang eines Zeichens. Seitenausg. bei Fehlern.

-FINRECEIVE
Abschluß des Empfangs

-WAITFORINREQ
Der Treiber geht in den Ruhezustand und wartet auf einen Eingabe-Request (und damit auf einen Puffer, in den die empfangenen Zeichen abgelegt werden können).
Seitenausgang, wenn ein anderes Ereignis vorher eintritt.

-INITOUTBUFF
Initialisierung des Ausgabe-Puffers, d.h. setze einen Zeiger auf das erste auszugebende Zeichen in diesem Puffer.

-FETCH
Hole das nächste Zeichen aus dem Ausgabe-Puffer und setze den Zeiger ein Zeichen weiter

-INITINBUFF
Initialisiere den Eingabe-Puffer, d.h. setze einen Zeiger an den Pufferanfang. Dort wird dann das erste Zeichen eingetragen.

-STORE
Speichere ein Zeichen in den Eingabepuffer und setze den Zeiger eine Zeichenposition weiter.

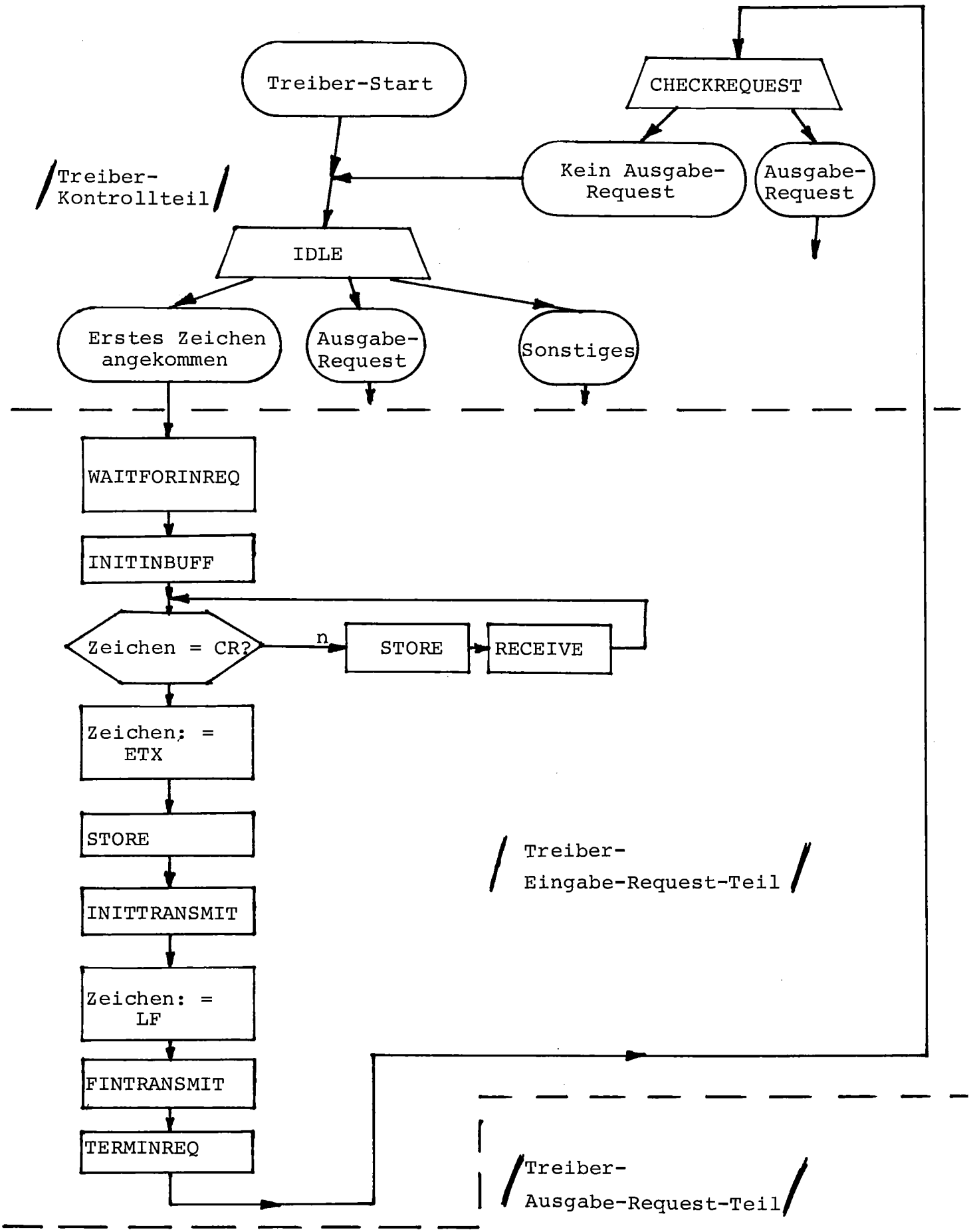
-TERMOUTREQ
Beende einen Ausgabe-Request, d.h. teile dem ausgebenden Prozeß mit, daß die Ausgabe der betr. Nachricht beendet ist (erfolgreich oder nicht erfolgreich - dann Übergabe von Fehleranzeigen)

-TERMINREQ
Beende einen Eingabe-Request, d.h. teile dem einlesenden Prozeß mit, daß die Eingabe der betr. Nachricht (erfolgreich oder nicht erfolgreich) beendet ist.

-GETNEXTPORT
Schalte um auf den nächsten Port

-CHECKREQUEST
Prüfe, ob an dem ausgewählten Port ein Ausgabe-,Eingabe-Request oder nichts ansteht und verzweige entsprechend.

5. Beispiel für ein Treiber-Programm



Jahrestagung
Siemens-Prozeßrechner-
Anwenderkreis
(vom 5.-7. April 1978)

SAK 78

Manuskript zum Vortrag

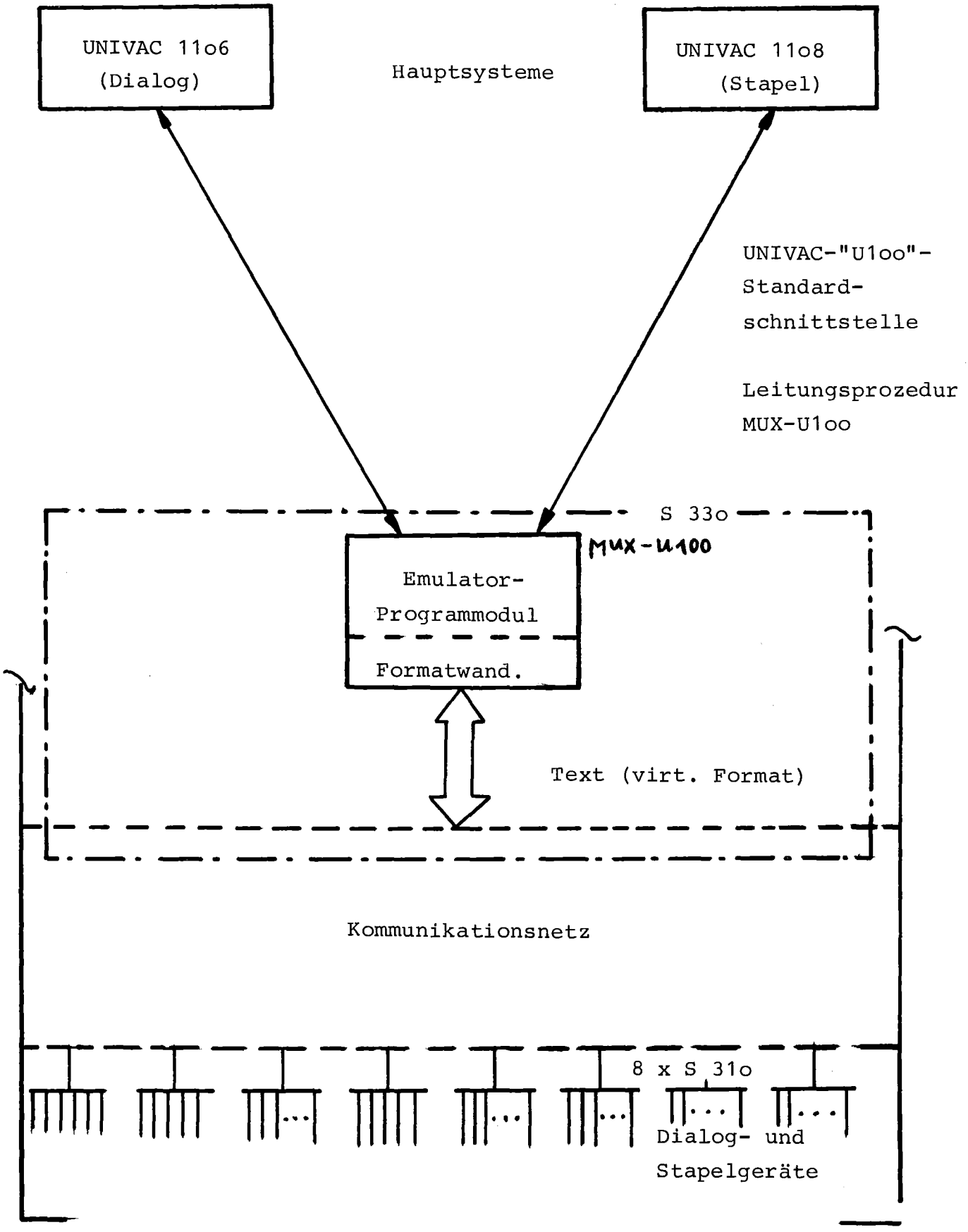
Implementierungsverfahren für Emulatoren von Leitungsmultiplexern am Beispiel eines UNIVAC-Terminalmultiplexers (MUX-U1oo)

von

Dipl.-Ing. Dietrich Eckert
Rechenzentrum
Universität Karlsruhe

Implementierungsverfahren für Emulatoren von Leitungsmultiplexern am Beispiel eines UNIVAC-Terminalmultiplexers (MUX-U1oo)

Um den Anschluß weitgehend beliebiger Terminals an vorgegebene UNIVAC-Rechenanlagen zu erreichen, wurde auf einer Siemens S 33o auf Anwenderebene ein Software-Emulator implementiert. Das Verfahren basiert auf einer Transitionssteuerung mit zugehöriger Prozedursteuermatrix, die aus einem gerichteten Graphen gewonnen wird.



Einordnung MUX-U 100 in Gesamtsystem

Emulator = inf. erk. und -verarbeitende Maschine

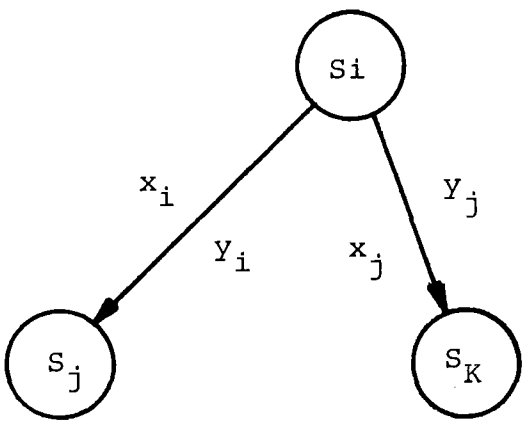
Beschreibung: durch Modell eines endlichen, determinist. Automaten z. B. "Mealy"-Automat

Darstellung durch:

- gerichtete Graphen
- Automatentafeln
- Entscheidungstabellen

Ergebnis: Aus Ablaufbeispielen für Leitungsprozedur (S.38) wird ein "gerichteter Graph" entwickelt. (S.39)

Beispiel:



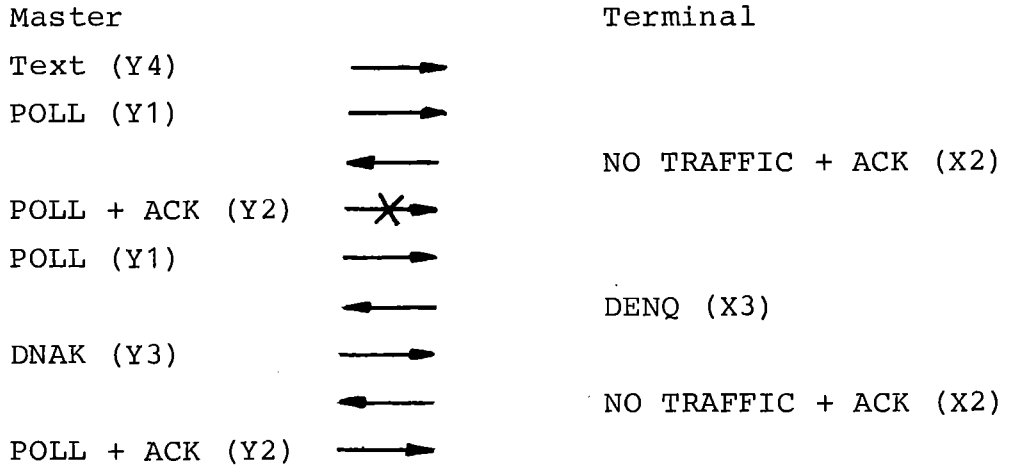
Zustandstripel:

S = Zustand
x = Ereignis
y = Aktion

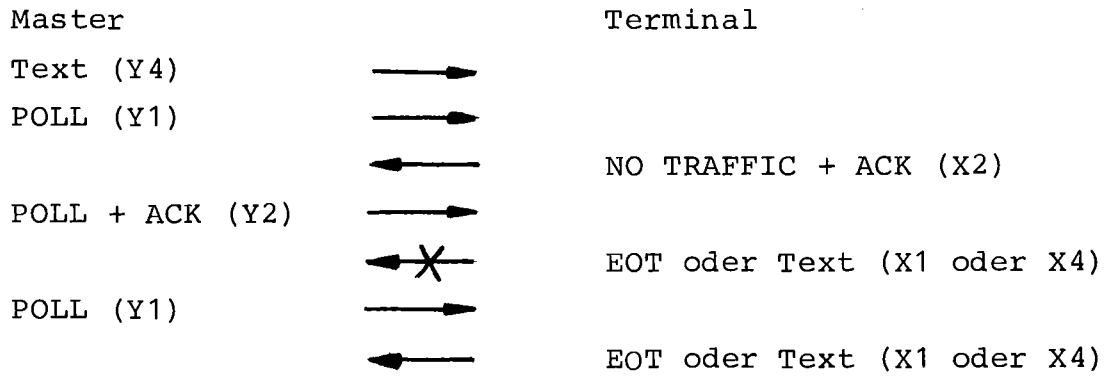
Aus Zustand 'S_i' erfolgt bei Auftreten des Ereignisses 'x_i' ein Zustandswechsel nach 'S_j' und die Aktion 'y_i' wird ausgeführt Transition 1).
Bei Auftreten von 'x_j' wird nach 'S_K' gewechselt und 'y_j' ausgeführt (Transition 2).

Auszug aus Ablaufbeispielen

Ablaufbeispiel



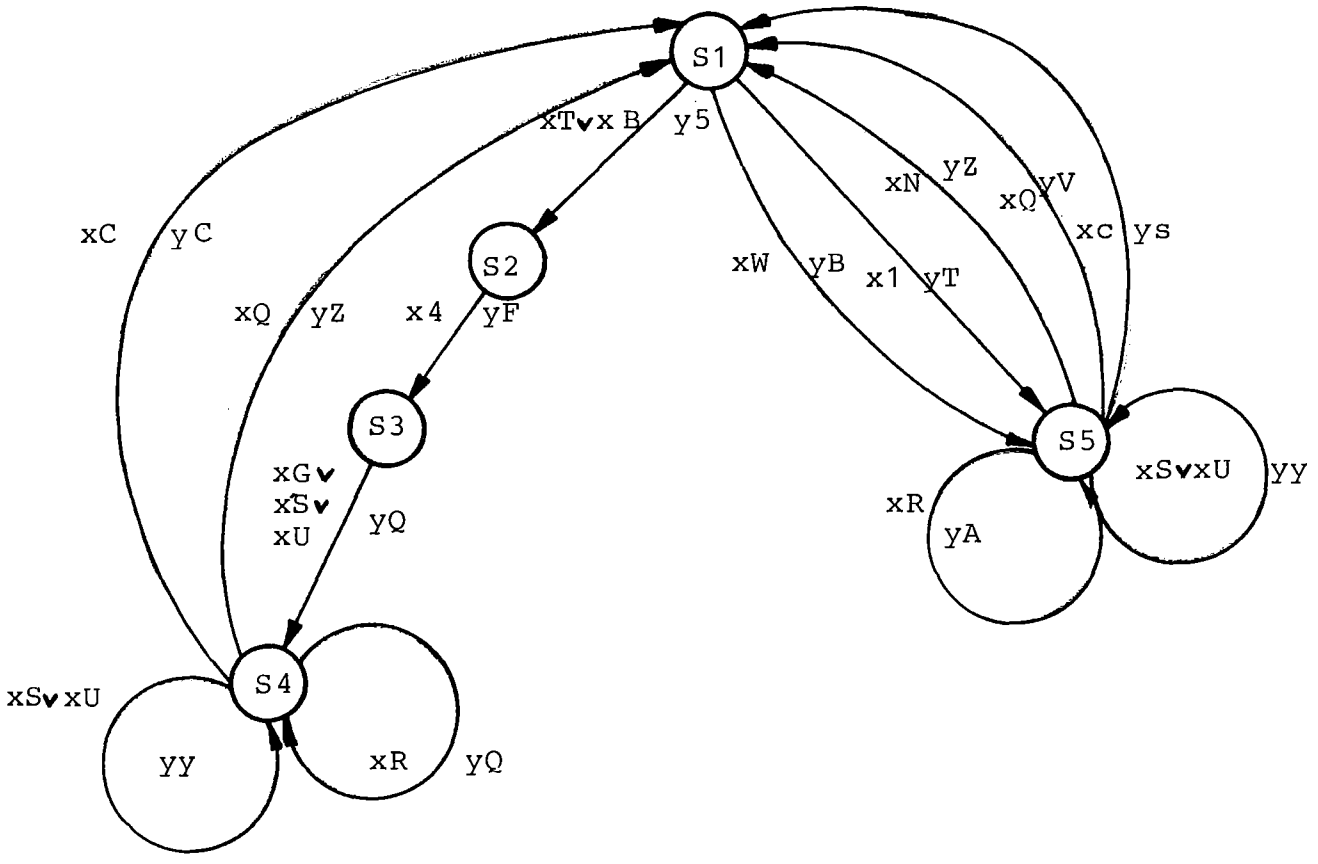
Ablaufbeispiel



Graph MUX-U1oo

(bildet zusammen mit verbaler Beschreibung
- der Ereignisse x_i
- und Aktionen y_i

eine exakte maschinenunabhängige Prozedurbeschreibung



Eingabeteil Ausgabeteil

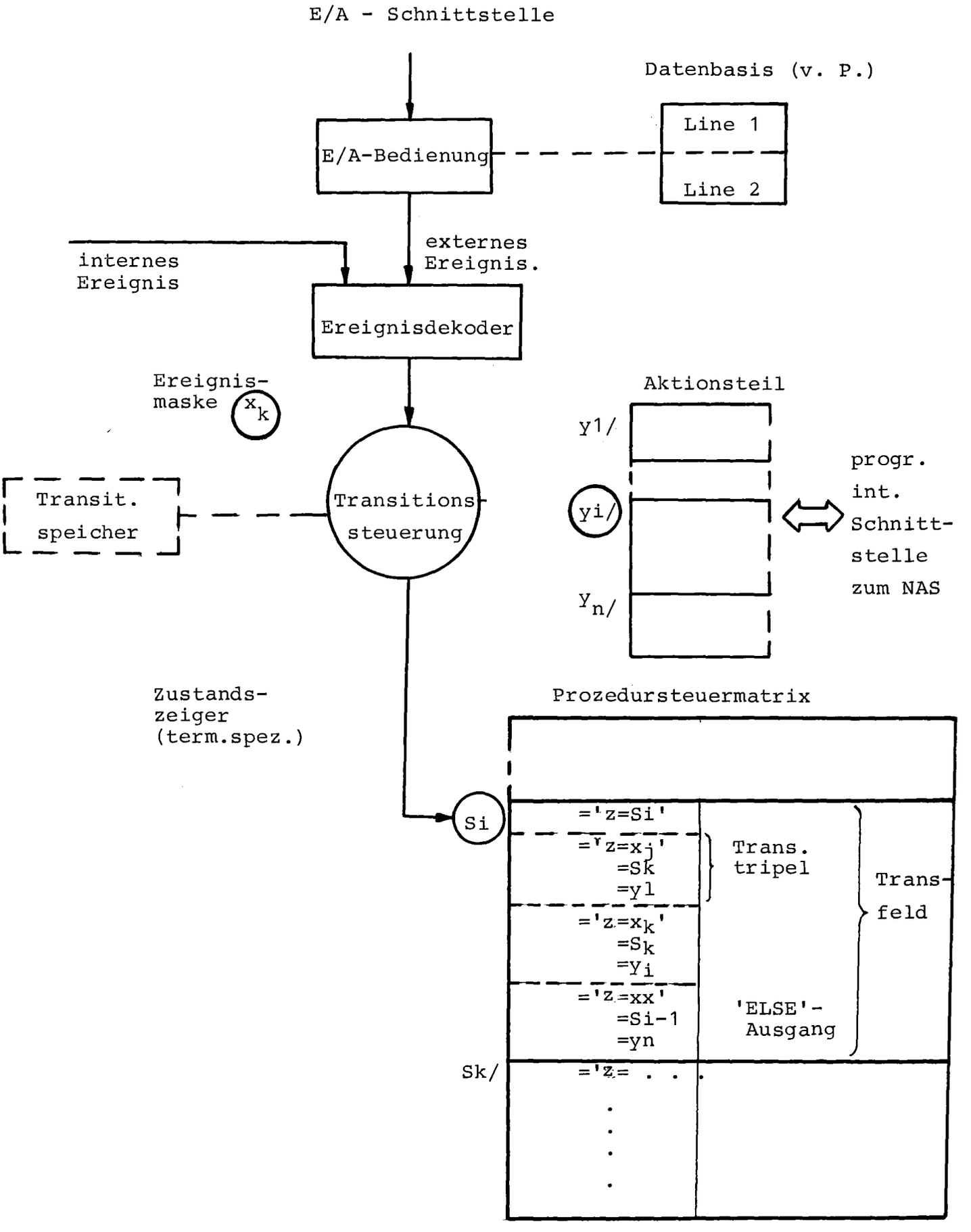
Einfache Umsetzbarkeit des Graphen in die entsprechende "Prozedursteuermatrix" (nächste Seite)

SIEMENS SYSTEM 300

SIEMENS 300: CROSS-MAKROPR. B7700 (03)

```
160      0    0                    =S2
161      0    0                  ='7=XX'
162      0    0                    =Y3
163      0    0                    =S2
164      0    0      S3/           ='7=X1'
165      0    0                    =YT
166      0    0                    =S4
167      0    0                  ='7=XN'
168      0    0                    =Y0
169      0    0                    =S5
170      0    0                  ='7=XT'
171      0    0                    =YX
172      0    0                    =S3
173      0    0                  ='7=XB'
174      0    0                    =YX
175      0    0                    =S3
176      0    0                  ='7=XX'
177      0    0                    =Y3
178      0    0                    =S3
179      0    0      S4/           ='7=X2'
180      0    0                    =Y3
181      0    0                    =S6
182      0    0                  ='7=XT'
183      0    0                    =YX
184      0    0                    =S4
185      0    0                  ='7=XB'
186      0    0                    =YX
187      0    0                    =S4
188      0    0                  ='7=XX'
189      0    0                    =Y3
190      0    0                    =S4
191      0    0      S5/           ='7=X2'
192      0    0                    =Y3
193      0    0                    =S1
194      0    0                  ='7=XT'
195      0    0                    =YX
196      0    0                    =S5
197      0    0                  ='7=XB'
198      0    0                    =YX
199      0    0                    =S5
200      0    0                  ='7=XX'
201      0    0                    =Y3
202      0    0                    =S5
203      0    0      S6/           ='7=X0'
204      0    0                    =Y0
205      0    0                    =S5
206      0    0                  ='7=XG'
207      0    0                    =YY
208      0    0                    =S7
209      0    0                  ='7=XS'
210      0    0                    =YY
211      0    0                    =S7
212      0    0                  ='7=XU'     *** 6.10.77
```

Auszug aus Prozedursteuermatrix



Programmaufbau MUX-U1oo

Erläuterung zum Programmaufbau MUX-U1oo

E/A-Bedienung:

Über E/A-Schnittstelle werden Ein- und Ausgaben abgewickelt. Eintreffende Nachrichten bestimmten Nachrichtenformats entsprechen externen Ereignissen.

Ereignisdekoder:

Externen und internen Ereignissen(z. B. Sonderbedingungen im Programmablauf) werden entsprechende Ereignismasken z. B. Zeichenfolge 'xk' zugewiesen.

Aktionsteil:

Enthält alle Aktionen (Programmablaufteile z. B. Textausgabe) mit den Einsprungmarken y;/

Prozedursteuermatrix:

Enthält 'Transitionsfelder' mit 'Transitionstripeln' (x, s, y).
Der letzte Tripel ist ein 'ELSE-Tripel' (vollständiger Graph), der den Fehlerausgang bildet.

Transitionssteuerung:

Die 'Transitionssteuerung' verwaltet terminalspezifische Zustandszeiger. Bei Auftreten eines 'Ereignisses x' übergibt der Ereignisdekoder der Transitionssteuerung eine entsprechende Maske 'xk'.

Die Transitionssteuerung durchsucht nun ab dem terminalspezifischen Zustandszeigerstand die Transitionstripel des zugehörigen Transitionsfeldes. Wird Identität der Ereignismasken erkannt, so wird die zugehörige Transition durchgeführt, d. h. der Zustandszeiger wird mit dem Folgezustand S (Einsprungmaske S_K/, d. h. Anfangsadresse des jeweiligen Transitionsfeldes) geladen und die 'Aktion' y (Einsprungmarke yi/) ausgeführt.

Transitionsspeicher:

Umlaufpuffer, in den alle ausgeführten Transitionen zwecks Rückverfolgbarkeit der prozeduralen Abläufe eingetragen werden (Prozedurablaufgedächtnis).

Vorteile des Implementierungsverfahrens

1. Einfache Umsetzbarkeit des Graphen in eine 'Prozedursteuermatrix'.

2. Exakte, maschinenunabhängige Prozedurbeschreibung.

3. Einfache Modifizierbarkeit der Prozedur durch Änderung der Matrix-Einträge (wire-wrap) auch 'online'.

4. Rückverfolgbarkeit der prozeduralen Abläufe durch Transitionsspeicher/ Zwang zu 'realtime'-Testverfahren wegen Prozedurverfälschung (vgl. Anhang)

5. Zustandsminimierung per Programm (vgl. Anhang)

ANHANG

```
SIEMENS 320/330  TEPOS-P DA4     BLATT:      2     DATUM:
V-PEG A-PEG A-HX:  .AN
13519 27855 6CCF:  .6^
13520 27856 6CD0:  .S1
13521 27857 6CD1:  .XT
13522 27858 6CD2:  .L6
13523 27859 6CD3:  .6^
13524 27860 6CD4:  .S2
13525 27861 6CD5:  .XP
13526 27862 6CD6:  .L6
13527 27863 6CD7:  .XG
13528 27864 6CD8:  .L6
13529 27865 6CD9:  .6^
13530 27866 6CDA:  .S3
13531 27867 6CDB:  .XG
13532 27868 6CDC:  .L6
13533 27869 6CDD:  .6^
13534 27870 6CDE:  .S4
13535 27871 6CDF:  .XC
13536 27872 6CE0:  .L6
13537 27873 6CE1:  .XG
13538 27874 6CE2:  .L6
13539 27875 6CE3:  .X0
13540 27876 6CE4:  .L6
13541 27877 6CE5:  .6^
13542 27878 6CE6:  .S1
13543 27879 6CE7:  .XT
13544 27880 6CE8:  .L6
13545 27881 6CE9:  .6^
13546 27882 6CEA:  .S2
13547 27883 6CEB:  .X4
13548 27884 6CEC:  .L6
13549 27885 6CED:  .XG
13550 27886 6CEE:  .L6
13551 27887 6CEF:  .6^
13552 27888 6CF0:  .S3
13553 27889 6CF1:  .XG
13554 27890 6CF2:  .L6
13555 27891 6CF3:  .6^
13556 27892 6CF4:  .S4
13557 27893 6CF5:  .XC
13558 27894 6CF6:  .L6
13559 27895 6CF7:  .XG
13560 27896 6CF8:  .L6
13561 27897 6CF9:  .X0
13562 27898 6CFA:  .L6
13563 27899 6CFB:  .XG
13564 27900 6CFC:  .L6
13565 27901 6CFD:  .X0
13566 27902 6CFE:  .L6
13567 27903 6CFF:  .XG
13568 27904 6D00:  .L6
13569 27905 6D01:  .X0
13570 27906 6D02:  .L6
13571 27907 6D03:  .XG
13572 27908 6D04:  .L6
13573 27909 6D05:  .X0
13574 27910 6D06:  .L6
13575 27911 6D07:  .6Z
13576 27912 6D08:  .S1
13577 27913 6D09:  .XT
```

Beispiel

Das an Leitung 'L6' (UNIVAC 1106) hängende Terminal '6^' erhält im Zustand 'S1' eine Textnachricht 'XT'

Auszug aus Transitionsspeicher

Reduzierte Schaltwerktafel (spaltenweise)

Zahl der Zustände = 9
0 = Redundanz

ZS:	1	2	3	4	5	6	7	8	9
EV: X1									
FZS:	2	0	2	0	1	7	8	0	0
YT1:	5	0	1	0	4	7	5	0	0
EV: X2									
FZS:	2	6	1	0	1	7	8	0	0
YT1:	5	0	4	0	4	7	5	0	0
EV: X3									
FZS:	2	4	1	4	1	7	8	9	9
YT1:	5	5	4	5	4	7	5	5	5
EV: X4									
FZS:	2	1	1	4	1	7	8	8	8
YT1:	5	4	4	2	4	7	5	3	2
EV: X5									
FZS:	2	3	1	4	1	7	8	8	8
YT1:	5	o	4	2	4	7	5	3	2
EV: X6									
FZS:	2	1	1	5	1	7	8	0	3
YT1:	5	4	4	6	4	7	5	0	0

Zustandsminimierung (Speicherplatz Laufzeit)

Eingelesene Schaltwerkstafel (spaltenweise)

0 = Redundanz

	ZS:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EV: X1																
	FZS:	2	0	4	0	0	7	0	0	0	11	12	0	0	0	0
	YT1:	5	0	1	0	0	4	0	0	0	7	5	0	0	0	0
EV: X2																
	FZS:	2	10	7	0	0	7	0	0	0	11	12	0	0	0	0
	YT1:	5	0	4	0	0	4	0	0	0	7	5	0	0	0	0
EV: X3																
	FZS:	2	0	7	5	0	7	2	9	0	11	12	0	14	0	14
	YT1:	5	0	4	5	0	4	5	5	0	7	5	0	5	0	5
EV: X4																
	FZS:	2	7	7	0	8	7	0	0	8	11	12	13	0	13	0
	YT1:	5	4	4	0	2	4	0	0	2	7	5	3	0	2	0
EV: X5																
	FZS:	2	3	7	0	8	7	0	0	8	11	12	13	0	13	0
	YT1:	5	0	4	0	2	4	0	0	2	7	5	3	0	2	0
EV: X6																
	FZS:	2	7	7	0	6	7	0	0	6	11	12	0	0	3	0
	YT1:	5	4	4	0	6	4	0	0	6	7	5	0	0	0	0
EV: X7																
	FZS:	2	7	7	0	8	7	0	0	8	11	12	13	0	13	0
	YT1:	5	4	4	0	2	4	0	0	2	7	5	3	0	2	0
EV: X8																
	FZS:	2	7	7	0	6	7	0	0	6	11	12	0	0	7	0
	YT1:	5	4	4	0	6	4	0	0	6	7	5	0	0	4	0
EV: X9																
	FZS:	2	7	7	0	0	7	0	0	4	11	12	0	0	15	0
	YT1:	5	4	4	0	0	4	0	0	1	7	5	0	0	3	0
EV: X1o																
	FZS:	2	0	7	0	0	7	0	0	0	2	12	0	0	0	0
	YT1:	5	0	4	0	0	4	0	0	0	5	5	0	0	0	0

Mehrfachimplementierung eines Reaktorschutzprogramms in PHI2, PASCAL und IFTRAN auf der SIEMENS 330

L. Gmeiner, W. Lemperle, U. Voges

Kernforschungszentrum Karlsruhe GmbH
Institut für Datenverarbeitung in der Technik
7500 Karlsruhe, Postfach 3640

1. Einleitung

In sicherheitsrelevanten Rechneranwendungen genügt es nicht, einen einzelnen Computer einzusetzen. Vielmehr ist aus Zuverlässigkeits- und Verfügbarkeitsgründen meist ein Mehrrechnersystem notwendig. Dies gilt auch für den im folgenden beschriebenen Fall eines Reaktorschutzsystems. Im zweiten und dritten Kapitel wird die Hardwarestruktur des Schutzsystems und die daraus abgeleiteten Softwareanforderungen erklärt. Im ursprünglichen Schutzsystem ist eine identische Software für alle drei Rechner vorgesehen. Um die Auswirkungen einer diversitären Programmierung der drei Rechner zu untersuchen, wurde eine Implementierung der Schutzprogramme in den Sprachen IFTRAN, PASCAL und PHI2 vorgenommen. Diese drei Implementierungen und die Erfahrungen, die dabei gemacht wurden, werden beschrieben. In Kapitel 4 wird eine kurze Bewertung und Zusammenfassung dieser experimentellen Untersuchung gegeben.

2. Hardware - Konfiguration eines Schutzsystems

Die Hardwarestruktur des zugrundeliegenden Reaktorschutzsystems ist in Bild 1 dargestellt. Die drei Rechner führen alle dieselben Berechnungen durch. Der Beginn eines neuen Berechnungszyklus wird durch einen Zeitgeber festgelegt. Die Temperaturkennwerte werden von den Thermoelementen übernommen und in den einzelnen Rechnern verarbeitet. Die Ergebnisse der Verarbeitung werden zwischen den Rechnern ausgetauscht und teilweise auf der Konsole im Kontrollraum ausgegeben. Das Abschalt- bzw. Freigabesignal eines jeden Rechners wird durch eine Hardwareschaltung einer 2 von 3-Bewertung unterworfen, die das endgültige Auslösesignal (abschalten oder nicht abschalten) erzeugt.

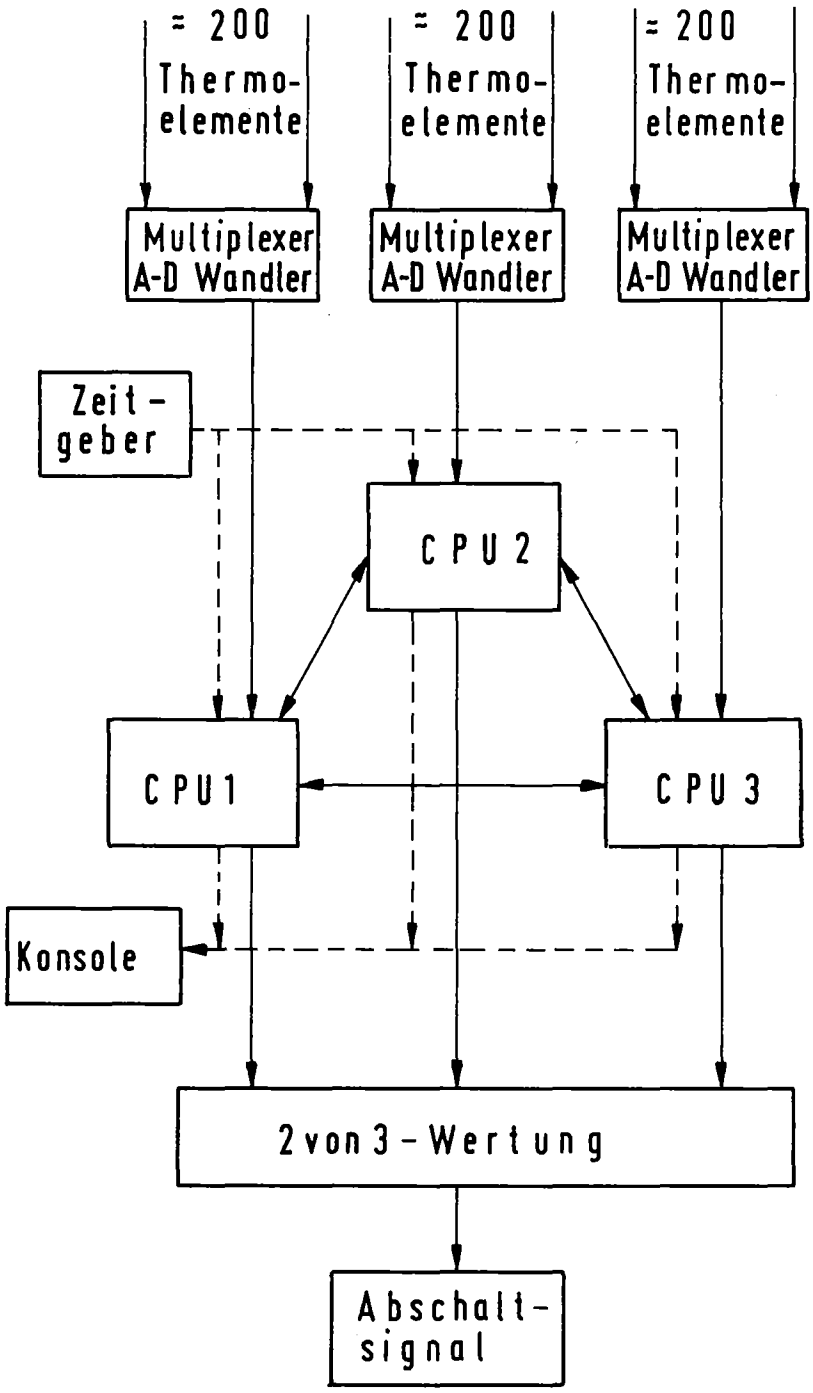


Bild 1: Hardware Konfiguration eines Reaktorschutzsystems

3. Diversitäre Programmierung

Für die drei oben beschriebenen Rechner ist im ursprünglichen Schutzsystem aus Kostengründen eine identische Software vorgesehen. Die anfallenden Kosten setzen sich zusammen aus Softwareerstellungskosten und Verifikationskosten für eben diese Software. Je höher die Zuverlässigkeitsanforderungen an eine solche Software sind, desto stärker überwiegen diese Verifikationskosten. Ein alternativer Ansatz zur identischen Software mit extrem hoher Zuverlässigkeit wäre die Verwendung der diversitären Programmierung. Hierbei wird auf jedem Rechner das Problem mit Hilfe eines anderen Algorithmus bzw. einer anderen Programmiersprache gelöst und jeder Rechner wird von einem anderen Team programmiert. Ein Vorteil dieser Lösung sind die geringeren Kosten. Da die einzelnen Programme keinen so hohen Zuverlässigkeitsanforderungen genügen müssen, sind sie billiger zu erstellen und zu prüfen. Die notwendige hohe Gesamtzuverlässigkeit wird dann durch die Diversität der Programme in den einzelnen Rechnern erreicht. Die Möglichkeit gleichartiger Fehler in allen drei Rechnerprogrammen wird durch die Diversität ebenfalls äußerst unwahrscheinlich.

Man unterscheidet zwei Arten von diversitärer Programmierung:

- verschiedene Implementierungen eines Algorithmus in einer einzigen Sprache
- verschiedene Implementierungen eines Algorithmus in verschiedenen Sprachen.

Während die erste Art der diversitären Programmierung einem Experiment von Avizienis /Avi 77/ zugrundeliegt, wird die zweite Art im folgenden beschrieben. Für die Implementierung ausgewählt wurden die Sprachen IFTRAN, PASCAL und PHI2 . Die Auswahl dieser Sprachen erfolgt zum einen aufgrund ihrer Verfügbarkeit auf der zugrundeliegenden Maschine SIEMENS 330, zum anderen können mit diesen Sprachen relativ einfach die für dieses Projekt ausgearbeiteten Programmierrichtlinien /Vog 75/ befolgt werden. Wir werden im folgenden kurz auf die einzelnen Implementierungssprachen eingehen.

3.1 Die Implementierungssprachen PASCAL, IFTRAN und PHI2

Ausgangspunkt für die einzelnen Implementierungen war eine gemeinsame Spezifikation. Um eine möglichst eindeutige, konsistente und vollständige Spezifikation zu gewährleisten, wurde eine formale Spezifikationsmethode gewählt. Die Spezifikation erfolgt dabei nach dem Input-Process-Output Ansatz /Boe 74/. Das gesamte zu spezifizierende Programmsystem besteht aus einer Menge von wechselseitig verknüpften Prozessen. Jeder dieser Prozesse besteht aus Input, Output und Zuständen, wobei ein Prozesszustand durch die Werte einer zugrundeliegenden Variablenmenge gegeben ist. Es zeigte sich, daß eine solche formale Spezifikation die Implementierungen in PASCAL, IFTRAN und PHI2 unterstützt und erleichtert. Es war vielfach möglich, die formale Spezifikation in einer quasi "natürlichen" Weise in den Code der jeweiligen Sprache zu übersetzen. Mit allen drei Implementierungssprachen lagen am Institut nur geringe Erfahrungen vor, so daß dieses Experiment auch als Eignungstest für diese Sprachen angesehen werden kann..

Die Sprache PASCAL /Jen 74/ bietet den Vorteil eines klaren Programmaufbaus mit globalen Daten, lokalen Daten, expliziten Typdeklarationen und sehr strengen Reglementierungen bezüglich der Kontrollstruktur. Im Verlauf der Implementierung zeigte es sich, daß der zugrundeliegende PASCAL-Compiler sehr gute Fehlermeldungen liefert und über gute Wiederaufsetzpunkte verfügt, so daß auch nachfolgende Fehlermeldungen noch viel Information bieten. Negativ ist in diesem Zusammenhang anzumerken, daß die vorliegende PASCAL-Version gegenüber Full-PASCAL eingeschränkt ist. Beispielsweise sind folgende Sprachfeatures nicht implementiert:

- ROUND-Funktion
- Explizite Artanpassung war nötig
  (Integer → Real)
- Die Definition von neuen Datentypen (z.B. Bereiche der natürlichen Zahlen) ist zwar syntaktisch möglich, wird aber zur Laufzeit nicht überprüft.

Eine weitere Unzulänglichkeit ist die mangelhafte E/A-Fähigkeit sowie die Fehler, die im PASCAL-System selbst entdeckt wurden. Durch die interpretative Abarbeitung ergibt sich zwangsläufig eine größere Laufzeit als bei vergleichbaren Compilern.

IFTRAN ist ein Preprozessor für strukturiertes FORTRAN /IFT 76/. Der Einsatz eines solchen Preprozessors bietet insbesondere den Vorteil einer klaren Kontrollstruktur (if-then-else, case, ...) sowie einen Mechanismus, mit dem COMMON-Blöcke nur einmal definiert werden (z.B. in BLOCK DATA) und durch Angabe ihrer Kurzaufrufe in den entsprechenden SUBROUTINEs identisch eingefügt werden. Außerdem ist das Programm durch automatisches Einrücken besser lesbar und damit auch wesentlich änderungsfreundlicher als vergleichbare FORTRAN-Programme.

Nachteilig wirkt sich bei allen Preprozessoren aus, daß ein zusätzlicher Übersetzungslauf nötig ist und daß eine geringfügige Codeerweiterung und Laufzeitverlängerung in Kauf genommen werden muß.

PHI2 ist ein um Strukturmakros erweiterter Assembler /PHI 77/. Hierdurch kann der Assemblerprogrammierer die gängigen Strukturierungselemente verwenden. Im folgenden sind einige aufgezählt:

$BEGIN...SBEND $IF...$THEN...$ELSE...$BEND $CASE...$OF...$OFREST...$BEND $WHILE...$DO...$BEND $CYCLE...$WHEN...$BREAK...$BEND	Makroaufrufe für Strukturblöcke
$ENTRY...$END $PASS $EXIT	Makroaufrufe für Prozeduren
$AND $OR $TOR	Bedingungs-verknüpfungen

Da mit PHI2 noch keine Erfahrungen vorlagen, ergaben sich gewisse Schwierigkeiten bei der Anwendung. Insbesondere mußten zunächst Fehler im PHI2-

System selbst beseitigt werden. Im Verlauf der Benutzung zeigten sich folgende Vor- und Nachteile:

Vorteile von PHI2

- Vorteile der strukturierten Programmierung auf Assemblerebene;
- Einfache Umsetzung von Strukturdiagrammen in Code;
- Allgemein bessere Lesbarkeit der Struktur (Verständlichkeit);
- Geringere Fehleranfälligkeit bezüglich der Struktur bei logischen Verschachtelungen und Verzweigungen.

Nachteile von PHI2

- Programm wird weder Speicher-noch Laufzeit-optimal;
- Längere Umsetzungszeiten durch zusätzliche Makroübersetzung (45 min Makroübersetzung, 15 min Assemblierung);
- Zwei Listings: Makro- und Assemblerlisting.

3.2 Erfahrungen bei der Implementierung

Wie erwartet, zeigte es sich, daß eine "natürliche" Transformation zwischen der formalen Spezifikation und dem Code der drei Sprachen möglich war. Es ergaben sich Schwierigkeiten aufgrund der mangelnden Betriebserfahrung und der Fehler, die in PASCAL und PHI2 auftraten.

Nachdem die Implementierung und der Vortest des Programmes vom jeweiligen Programmersteller durchgeführt war, wurde jedes Programm noch-

mals von einer unabhängigen Person auf im voraus festgelegte Testkriterien hin untersucht. Ein solches Testkriterium war das Durchlaufen aller Verzweigungen. Dieses Kriterium klingt zwar sehr einfach und wenig aussagefähig, es zeigte sich jedoch, daß damit wichtige Testfälle abgedeckt wurden, die beispielsweise im Vortest des Programmierers nicht in Erwägung gezogen wurden. Aufgrund dieses Kriteriums wurden einige Fehler und Programmanomalitäten, wie nicht erreichbarer Code, entdeckt.

Ausgangspunkt für die Testdatenerzeugung war die Programmstruktur (Verzweigungsbedingungen etc.), während die zugehörigen Testresultate aus der Spezifikation abgeleitet wurden. Insbesondere beim Test der IFTRAN-Implementierung war der Einsatz der Testsysteme RXVP und SADAT /RXV 75/, /Ams 77/, /Sei 76/ hilfreich. Ein weiteres wertvolles Hilfsmittel war ein automatisches Ergebnisvergleichsprogramm. Dieses Programm versorgt jede der drei Implementierungen mit identischen Eingangsdaten und überprüft die anfallenden Zwischen- und Endergebnisse. Falls Abweichungen in einer der drei Implementierungen auftreten, wird eine entsprechende Meldung gemacht. Aufgrund der vielen Eingangs- und Zwischendaten erwies sich das automatische Vergleichsprogramm als sehr hilfreich, weil dadurch eine fehlerbehaftete manuelle Überprüfung überflüssig wurde. Aufgrund der diversitären Implementierungen in IFTRAN, PASCAL und PHI2 wurden Mehrdeutigkeiten der Spezifikation entdeckt, die zu unterschiedlichen Implementierungen führten. Diese Unterschiede wurden mit Hilfe des automatischen Vergleichsprogramms entdeckt.

In der folgenden Tabelle werden einige Werte von charakteristischen Programmgrößen dargestellt.

	IFTRAN	PASCAL	PHI2
Codegröße (Worte)	21251	13463	14205
Programmzeilen ohne Kommentar	1077	783	1606
Laufzeit normiert auf PHI2	1,21	3,65	1

Bezüglich der Codegröße ist das günstige Abschneiden von PASCAL interessant. Im Falle der Laufzeit hingegen tritt der interpretative Charakter von PASCAL deutlich zu Tage. Die Laufzeit des PASCAL-Programmes ist ungefähr 4 mal größer als bei PHI2 . Interessant ist auch, daß IFTRAN relativ viel mehr Code erzeugt als die restlichen beiden Sprachen. Bei der Laufzeit hingegen schneidet IFTRAN wesentlich günstiger ab und ist nahezu so schnell wie PHI2 .

4. Zusammenfassung

Aufgrund der hier gewonnenen Erfahrungen scheint die diversitäre Programmierung ein angemessenes Mittel zu sein, um einerseits identische Programmfehler in allen drei Rechnern zu verhindern, andererseits Mehrdeutigkeiten der Spezifikation aufzudecken. Während des gesamten Experiments wurden die auftretenden Fehler detailliert aufgezeichnet und werden im Augenblick ausgewertet. Wir hoffen, durch diese Auswertung genaue Hinweise auf die Schwachstellen der diversitären Programmierung zu bekommen, um diese Erfahrung dann in späteren Projekten einsetzen zu können.

5. Referenzen

/Ams 77/ Amschler, A., Gmeiner, L.
SADAT 2 - Ein System zur automatischen Vorbereitung, Durchführung und Auswertung von Tests für FORTRAN-Programme
KfK-Ext. 13/77-2, Oktober 1977

/Avi 77/ Avizienis, A., Chen, L.
On the Implementation of N-Version Programming for Software Fault-Tolerance during Program Execution
Compsac, Chicago, October 1977

/Boe 74/ Boehm, B.W.
Some Steps toward Formal and Automated Aids to Software Requirements Analysis and Design
2nd IFIP Congress, Stockholm, 1974

/IFT 76/ IFTRAN: Structured Programming Preprocessors for FORTRAN
General Research Corp., Santa Barbara, 1976

/Jen 74/ Jensen, K., Wirth, W.
PASCAL - User Manual and Report
Lecture Notes in Computer Science, Springer, 18, 1974

/PHI 77/ Programmierhilfe-Makros für strukturierte Programmierung
SIEMENS Programmbeschreibung P71100-J1015-X-X-35

/RXV 75/ RXVP, FORTRAN automated Verification System
General Research Corp., Santa Barbara, 1975

/Sei 76/ Seifert, M.
SADAT - Ein System zur automatischen Durchführung und Auswertung von Tests
KfK-Ext. 13/75-5, Juni 1976

/Vog 75/ Voges, U., Ehrenberger, W.
Vorschläge zu Programmierrichtlinien für ein Reaktorschutzrechnersystem
KfK-Ext. 13/75-2, Kernforschungszentrum Karlsruhe, 1975

Analyse, Test und Dokumentation von FORTRAN-Programmen mit dem automatischen Testsystem RXVP/S330

W. Geiger

Kernforschungszentrum Karlsruhe
Institut für Datenverarbeitung in der Technik
7500 Karlsruhe, Postfach 3640

Kurzfassung:

Es wird das automatische Testsystem RXVP/S330 vorgestellt. Mit diesem System können PROZESS-FORTRAN 300-Programme analysiert und der Programmcode dokumentiert werden. Außerdem kann mit ihm die Ausführungshäufigkeit der einzelnen Programmteile bei den Testläufen analysiert werden.

In dieser Beschreibung werden die Fähigkeiten und die Arbeitsweise des Testsystems dargestellt. Weiter wird auf den Aufwand und die Erfahrungen beim Einsatz eingegangen.

1. Einleitung

Untersuchungen von B.W. Boehm an einigen Realzeit-Software-Systemen /Boe 73/ ergaben, daß ca. 50 % der Entwicklungskosten von Software-Systemen in der Testphase anfallen. Weitere Untersuchungen an kommerziellen Software-Systemen /Els 76, Boe 76/ zeigten, daß ca. 75 % der gesamten Arbeitszeit der Programmierer für die Wartung (inkl. Programmodifikationen) aufgewendet werden.

Um diese hohen Kosten für das Austesten und Warten von Programmsystemen zu reduzieren und ihre Zuverlässigkeit zu erhöhen, werden in den letzten Jahren zunehmend automatische Hilfsmittel entwickelt. Ein solches Hilfsmittel ist das von der Firma General Research Corporation entwickelte automatische Testsystem RXVP für FORTRAN- und IFTRAN-Quellprogramme /Gen 77a, Gen 77b/.

Dieses Testsystem wird im Institut für Datenverarbeitung in der Technik des Kernforschungszentrums Karlsruhe als Hilfsmittel eingesetzt, um Software für kerntechnische Anlagen auszutesten und dient als Grundlage für eigene Entwicklungen für den PDV-Einsatz.

2. Übersicht über das Testsystem RXVP

Das Testsystem RXVP geht von dem Quelltext des Testprogramms aus. (Mit 'Testprogramm' ist hier und im weiteren das zu testende Programm gemeint.) Es führt eine statische Analyse des Quellprogramms durch, bestehend aus einer lexikalischen Analyse und einer Kontrollstruktur-Analyse. Es generiert dann eine Reihe von Dokumentationsreports, auf die noch näher eingegangen wird.

Weiter liefert es ein quantitatives Maß für die Vollständigkeit des Testens, die mit gewissen Testdaten erreicht wird. Als Kriterium für vollständiges Testen wird dabei das Kriterium verwendet, das in Fachkreisen am meisten als adequates Testkriterium anerkannt ist und als 'minimal vollständiges Testen' /Hua 75/ oder 'vollständige Überdeckung' /Mil 77/ bezeichnet wird: Alle Verzweigungsalternativen im Programm sollen mindestens einmal durchlaufen werden.

Um automatisch den Prozentsatz der Verzweigungen zu bestimmen, die beim Testen durchlaufen werden, wird eine dynamische Analyse durchgeführt. Das Testsystem analysiert die Kontrollstruktur des Programms und instrumentiert

jede Verzweigungsalternative, d.h. fügt an jeder Verzweigungsalternative einen Aufruf an eine Datensammelroutine ein, bei dem als Parameter eine Verzweigungskennung übergeben wird. Das instrumentierte Programm wird dann mit der Datensammelroutine ausgeführt und anschließend die Ausführungshäufigkeit der einzelnen Verzweigungen analysiert.

3. Erweiterungen und Anpassungen zum Testsystem RXVP/S330

Das Testsystem RXVP verarbeitet Quellprogramme in ANSI-Standard-FORTRAN und IFTRAN (siehe auch /Gme 78/) und wurde bisher nur auf Großrechnern installiert. Im vergangenen Jahr haben wir es auf der IBM/370 des KfK installiert. Um es auch für S330-Benutzer verwendbar zu machen, wurde das Testsystem so erweitert, daß es die meisten der über den ANSI-Standard hinausgehenden Sprachelemente von PROZESS-FORTRAN 300 verarbeiten kann. Außerdem wurde der hintere Teil der dynamischen Analyse des Testsystems, die Datensammelroutine und die Ausführungshäufigkeits-Analyse, an die S330-Anlage des IDT angepaßt und auf ihr installiert.

In Tabelle 1 ist der vom erweiterten Testsystem verarbeitete Quellsprachumfang (unvollständig) aufgelistet. Das erweiterte Testsystem verarbeitet alle Sprachelemente von PROZESS-FORTRAN 300 mit Ausnahme von Rücksprungmarken als aktuellen Parametern beim Unterprogrammaufruf und von Einstreuungen von Assemblercode in das FORTRAN-Programm. Das erstere darf im Quelltext auftreten, wird aber bei der Kontrollstruktur des Programms nicht berücksichtigt.

4. Funktionen und Ablauf von RXVP/S330

Das Testsystem RXVP/S330 hat folgende Arbeitsweise: Auf der IBM/370 des KfK wird die statische Analyse des Testprogramms durchgeführt und die aufbereitete Information des Programms in einer wiederverwendbaren Datenbasis abgelegt (Abb.1). Ausgehend von der Datenbasis werden dann - gesteuert über RXVP-Kommandos - verschiedene Dokumentationsreports erzeugt.

Von allen Moduln wird ein entsprechend der Kontrollstruktur eingerücktes Quellprogramm-Listing ausgegeben, in dem die Anweisungsnummern und die Nummern der Verzweigungsalternativen (decision-to-decision-paths) definiert werden. In Abbildung 2a ist ein Beispiel für ein IFTRAN-Programm, in Abbildung 2b ein Beispiel für ein PROZESS-FORTRAN 300-Programm aufgeführt. In einem weiteren Listing werden die Verzweigungsnummern noch näher definiert, z.B. Verzweigung 4 ist der Wahr-Zweig, Verzweigung 5 der Falsch-Zweig der IF-Bedingung. Die hier definierten Nummern werden in späteren Listings verwendet.

Sprachelemente von PROZESS-FORTRAN 300	erlaubt
- FORTRAN IV - Sprachumfang inkl. REAL*8, COMPLEX*16 etc., IMPLICIT, DEFINE FILE ...	ja
- Binär- und Hexakonstanten z.B. BL'ØØ1', HR 'ØF'	ja
- Maskierungsoperationen, .EOR.	ja
- Schiebeoperationen .ISHIFT. und .LSHIFT.	ja
- DECODE, ENCODE	ja
- Rücksprungmarke als Unterprogramm-Parameter	(nein)
- Mischen von Assembler und FORTRAN	nein
- Systemfunktionen SYSTIME, SYSDATE...	ja
- EXECUTE	ja
- sämtliche Standardfunktionen	ja
- sämtliche Standardsubroutinen inkl. Bitmusterbearbeitung, Prozeß-Ein-/Ausgabe etc.	ja

Tabelle 1: Erlaubte Quellsprache von RXVP/S330

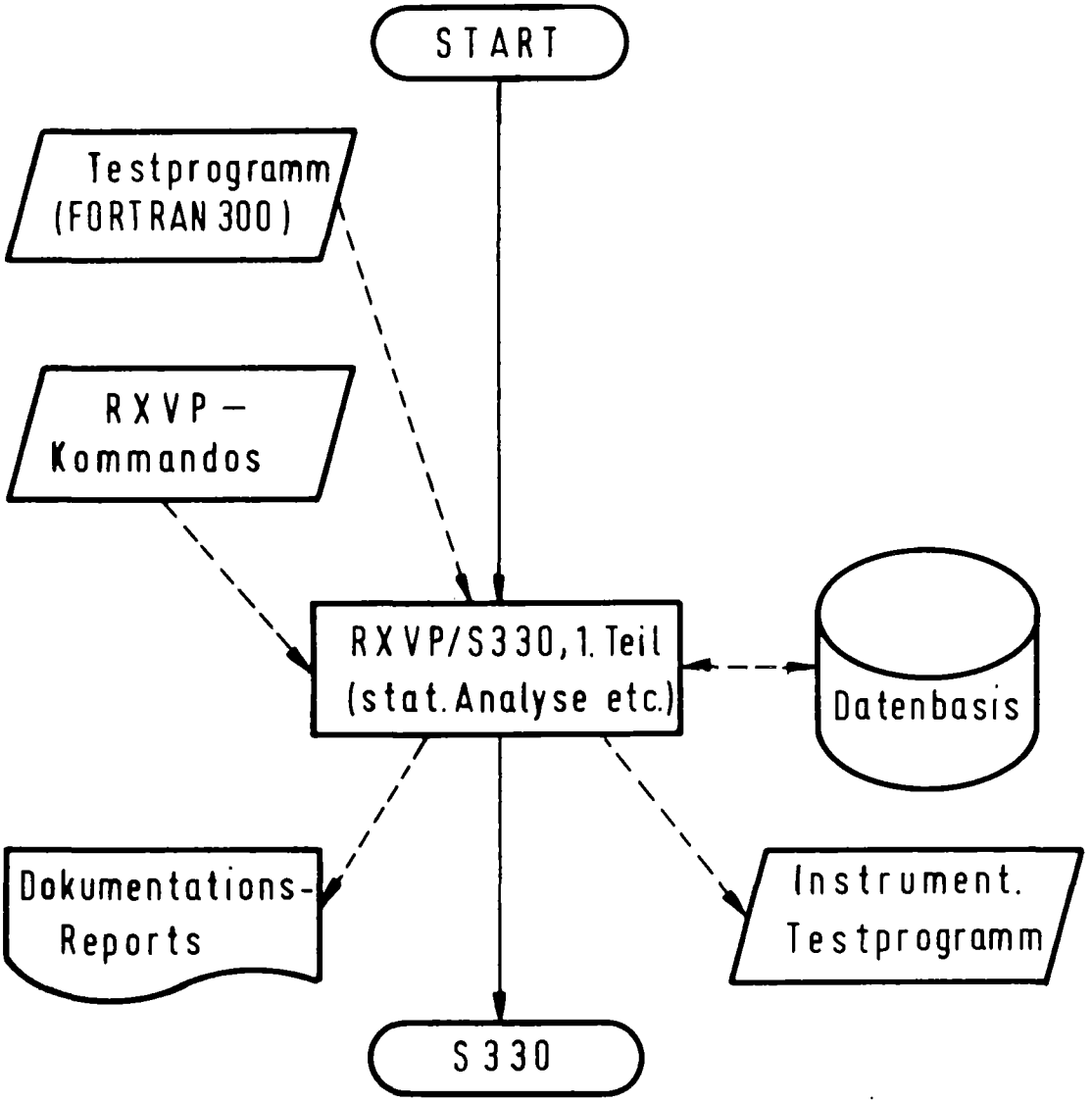


Abb. 1: Ablauf und Funktionen von RXVP/S330, 1. Teil
Ablauf auf der IBM/370

```
STATEMENT LISTING                SUBROUTINE FEST ( TI, TIN, BABI )                    PAGE     52

  NO. LEVEL    LABEL   STATEMENT TEXT...                                              DDPATHS
-------------------------------------------------------------------------------------------------
    1                  SUBROUTINE FEST ( TI, TIN, BABI )                             (  1)
    2          C       ******************************************** FEST         **
    3          C       FESTGRENZENPRUEFUNG UND MELDUNG
    4          C       PARAMETER
    5                  INTEGER*2 BABI ( 205 )
    6                  REAL*4 TI ( 205 ), TIN
    7          C       GLOBALE DATEN
    8                  COMMON / GRUPPE / GIE
    9                  INTEGER*2 GIE ( 205 )
   10                  COMMON / AUSGAB / ZNA, STA, IA, ERA, TPA
   11                  INTEGER*2 ZNA, STA, IA, ERA, TPA
   12                  COMMON / KON2 / KNORM, KGEST, KGEF, KWAHR, KFALS
   13                  INTEGER*2 KNORM, KGEST, KGEF, KWAHR, KFALS
   14          C       INTERNE DATEN
   15                  INTEGER*2 I
   16          C
   17                  CALL TRACE ( 10HFEST       )
   18                  FOR ( I = 1 TO 205 )                                          (  2-  3)
   19 ( 1)             .  IF ( GIE ( I ) .EQ. 5 )                                    (  4-  5)
   20 ( 2)             .  .  BABI ( I ) = KNORM
   21 ( 2)             .  .  CALL MELDNG ( ZNA, STA, I, 0, 0 )
   22 ( 2)             .  .  CALL BLIND
   23 ( 1)             .  ELSE
   24 ( 2)             .  .  IF ( TIN .LE. 650 )                                     (  6-  7)
   25 ( 3)             .  .  .  IF ( TI ( I ) .GT. 650 )                             (  8-  9)
   26 ( 4)             .  .  .  .  BABI ( I ) = KGEF
   27 ( 4)             .  .  .  .  CALL MELDNG ( ZNA, STA, I, 1, 4 )
   28 ( 3)             .  .  .  ORIF ( TI ( I ) .LT. TIN )                           ( 10- 11)
   29 ( 4)             .  .  .  .  BABI ( I ) = KGEST
   30 ( 4)             .  .  .  .  CALL MELDNG ( ZNA, STA, I, 1, 1 )
   31 ( 3)             .  .  .  ELSE
   32 ( 4)             .  .  .  .  BABI ( I ) = KNORM
   33 ( 3)             .  .  .  ENDIF
   34 ( 2)             .  .  ENDIF
   35 ( 1)             .  ENDIF
   36                  ENDFOR
   37                  CALL TRACE ( 10HFEST  ENDE )
   38                  RETURN
   39          C       ******************************************** FEST    ENDE **
   40                  END
-------------------------------------------------------------------------------------------------
```

Abb. 2a: Quellprogramm-Listing (IFTRAN-Programm)

```
STATEMENT LISTING                SUBROUTINE MELDPA ( NAME )                           PAGE      1

  NO. LEVEL    LABEL   STATEMENT TEXT...                                              DDPATHS
-------------------------------------------------------------------------------------------------
    1                  SUBROUTINE MELDPA ( NAME )                                    (  1)
    2          C       ************************************************
    3          C       BEISPIEL FUER PROZESS-FORTRAN 300-PROGRAMM     *
    4          C       ************************************************
    5                  IMPLICIT INTEGER ( A - Z )
    6                  INTEGER DATE ( 3 ), FELD ( 7 ), PNR ( 1 )
    7                  INTEGER*2 NAME ( 3 )
    8          C
    9                  MLB = HR'000F'
   10                  MPSP = BR'111000000000'
   11          C
   12                  TIME = SYSTIME
   13                  EXECUTE DATE ( DATE )
   14                  PNR = SYSPNUM
   15                  EXECUTE STTSK ( PNR ( 1 ), FELD, ANZ )
   16                  LB = FELD ( 4 ) .AND. MLB
   17                  PSP = FELD ( 4 ) .AND. MPSP
   18                  PSP = PSP .LSHIFT. - 8
   19                  WRITE ( 0, 1000 ) ( NAME ( I ), I = 1, 3 ), PNR, FELD ( 5 ), LB,
                      *PSP, ( DATE ( II ), II = 3, 1, - 1 ), TIME
   20          1000    FORMAT(1X,3A2,2X,'PROGR.NR.:',I4,2X,'PRIOR.:',I4,2X,'LAUFBER.:',
                      *I4,2X,'PROGR.SP.NR.:',I4,2X,'DATUM:',I2,'.',I2,'.',I4,2X,U12)
   21                  RETURN
   22                  END
-------------------------------------------------------------------------------------------------
```

Abb. 2b: Quellprogramm-Listing (PROZESS-FORTRAN 300-Programm)

Außerdem wird die Aufrufstruktur der Module analysiert und ausgegeben. Für alle Module werden in einem ersten Listing sämtliche Aufrufe von diesem zu anderen Moduln und von anderen zu diesem Modul aufgeführt (Abb. 3). In praxi ist vor allem das letztere vorteilhaft, da diese Information vom Compiler und Binder nicht direkt geliefert wird und vom Benutzer erst mühsam zusammengestellt werden muß. In einem zweiten Bericht wird die Aufrufstruktur über und unter dem betreffenden Modul bis zu einer Tiefe von fünf Schichten dargestellt (Abb. 4). Weiter werden die Modulaufrufe komprimiert in einer Aufrufmatrix dargestellt (Abb. 8). Mit diesen Berichten wird die Modulaufrufstruktur eines Programmsystems transparent gemacht. Diese Information ist notwendig, um einen Überblick über ein Programmsystem zu gewinnen, z.B. wenn es segmentiert werden soll.

Ferner wird eine globale Kreuzreferenzliste ausgegeben (Abb. 5). Darin wird neben dem Modulnamen und der Zeilennummer, in der die Symbole auftreten, die Art des Auftretens angegeben: ob die Variable definiert, gesetzt oder verwendet wird.

Schließlich wird noch der Datenfluß zwischen den Moduln über die COMMON-Blöcke analysiert und dokumentiert. In der COMMON-Block-Matrix (Abb. 6) wird angegeben, in welchen Moduln die einzelnen COMMON-Blöcke auftreten; in der COMMON-Symbole-Matrix (Abb. 7) wird für jede COMMON-Variable angegeben, ob sie in dem referenzierenden Modul nur gesetzt oder nur verwendet oder gesetzt und verwendet usw. wird.

Die Kreuzreferenzliste und die COMMON-Matrizen erweisen sich als sehr nützlich, wenn Änderungen an einem Programmsystem vorgenommen werden sollen, einmal um festzustellen,wo die Variablen gesetzt werden, außerdem um festzustellen,wo die betroffenen Variablen verwendet werden, um die Auswirkungen von Programmänderungen kontrollieren zu können.

Neben der Erzeugung dieser Dokumentationsreports wird auf der IBM/370 die Instrumentierung der Kontrollstruktur durchgeführt, siehe Kap. 2 und Abb.1. Das instrumentierte Testprogramm wird auf Magnetband oder Karten ausgegeben und zur S330 transportiert. Auf der S330 wird es eingelesen, mit dem FORTRAN-Compiler übersetzt und mit der als Grundsprachemodul vorliegenden Datensammelroutine gebunden (Abb. 9).

Das instrumentierte Programm wird dann mit gewissen Test-Eingabedaten ausgeführt. Neben der normalen Ausgabe (d.h. der Ausgabe des uninstrumentierten Programms) gibt das instrumentierte Programm auch Daten über die Ausführung der einzelnen Verzweigungsalternativen in eine Zwischendatei aus.

```
INVOCATION SPACE                    SUBROUTINE FEST ( TI, TIN, BABI )

----------------------------------------

INVOCATIONS FROM WITHIN THIS MODULE
----------------------------------------

MODULE BLIND
STMT =   22          CALL BLIND

MODULE MELDNG
STMT =   21          CALL MELDNG ( ZNA , STA , I , 0 , 0 )
STMT =   27          CALL MELDNG ( ZNA , STA , I , 1 , 4 )
STMT =   30          CALL MELDNG ( ZNA , STA , I , 1 , 1 )

MODULE TRACE
STMT =   17          CALL TRACE ( 10HFEST       )
STMT =   37          CALL TRACE ( 10HFEST  ENDE )

INVOCATIONS TO THIS MODULE FROM WITHIN LIBRARY
----------------------------------------

MODULE BERBIN
STMT =   12          CALL FEST ( TX , TALT , BRE )
STMT =   17          CALL FEST ( TY , TALT , BLI )

MODULE INIT
STMT =   31          CALL FEST ( DUR1 , DIN , B1 )
STMT =   61          CALL FEST ( DUR2 , DIN , B2 )
```

Abb. 3: Listing der Aufrufe von und zu einem Modul

```
VOCATION BANDS                  SUBROUTINE FEST ( TI, TIN, BABI )

 LEVEL 2

VEL      -5       -4       -3       -2       -1        0        1        2        3        4
-------------------------------------------------------------------------------------------
                                                      FEST
                                              BERBIN           BLIND
                                    MAIN                               TRACE
                                              INIT             MELDNG
                                    MAIN                               CNTRDR
                                                                       TRACE
                                                               TRACE
-------------------------------------------------------------------------------------------
```

Abb. 4: Modulaufrufstruktur

```
CROSS REFERENCE

GENERAL CROSS REFERENCE LISTING
SYMBOL     MODULE     USED/SET/DEFINITION ( * INDICATES SET, D INDICATES DEFINITION )
ABI        PRUEF       32D  58*  59   59
ABS        WANBIN      61
AE         BLOCKDAT    22D  23D
           EINMES      22D  23D  59*
           MAIN        19D  20D
           PAREIN       8D   9D
           PROTA       15D  16D  22   27   35   43   45
           PRUEF       13D  14D  63   63
AUSBIN     AUSBIN       1
           MAIN        41
AUSCRM     AUSCRM       1
           MAIN        54
AUSMLD     AUSMLD       1
           MAIN        60
           STOP        23
AUSPRT     AUSPRT       1
           INIT        40   41   42   43
           PROTA       26   34   42   44   46
BABI       FEST         1    5D  20*  26*  29*  32*
BERBIN     BERBIN       1
           MAIN        39
BERTI      BERBIN       8
           BERTI        1
             :
             :
```

Abb. 5: Globale Kreuzreferenzliste

COMMONS MATRIX

LIBRARY COMMON BLOCK MATRIX

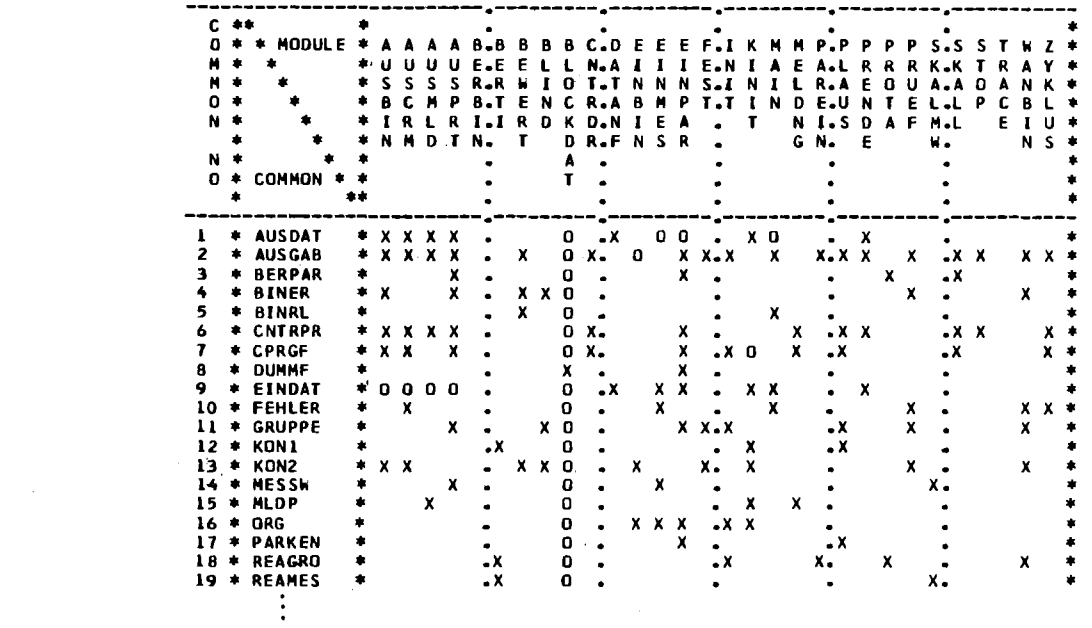
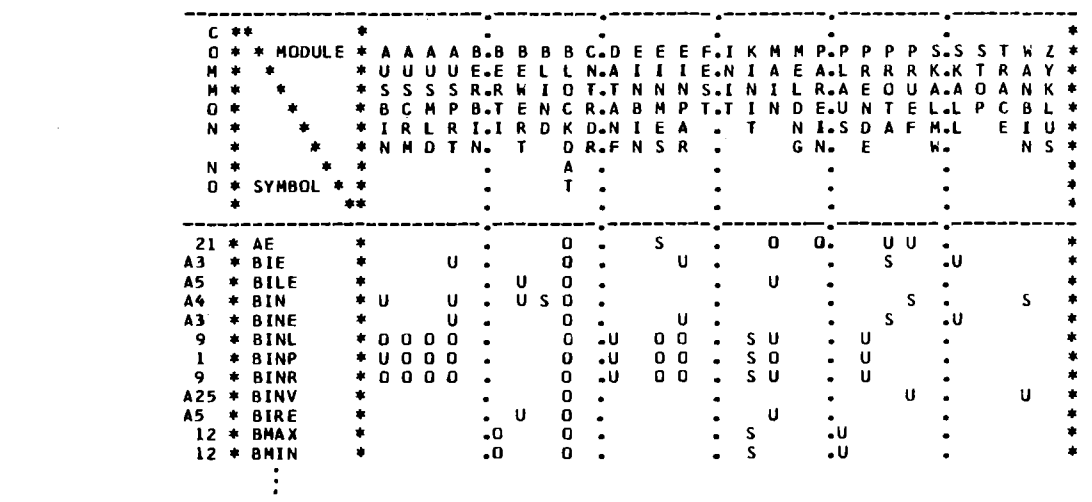


Abb. 6: COMMON-Block-Matrix

LIBRARY COMMON SYMBOL MATRIX (SUMMARY OUTPUT)



LEGEND

COMMONS VS. MODULES

X => AT LEAST ONE SYMBOL REFERENCED
O => NO SYMBOL EVER REFERENCED

SYMBOLS VS. MODULES

X => SYMBOL SET AND USED
O => SYMBOL NEVER SET OR USED
S => SYMBOL SET ONLY
U => SYMBOL USED ONLY
E => SYMBOL EQUIVALENCED (OVERLAID) ONLY
A => SYMBOL IS AN ARRAY

Abb. 7: COMMON-Symbole-Matrix

LIBRARY DEPENDENCE

```
*********************************************
** INVOKEE *                                * *
* *        *AAAABBBBBCDEEEFIKMMPPPPPSSSTwZ*A*
*  *       *UUUUEEELLNAIIIENIAEALRRRKKTRAY*B*
*   *      *SSSSRRwIOTTNNNSINILRAECUAAOANK*S*
*    *     *BCMPBTENCRABMPITTINDEUNTELLPCBL* *
*     *    *IRLRIIROKDNIEA  T NISDAFLM EIU* *
*      *   *NMDTN T DRFNSR    GN E   w  NS* *
*       *  *        A                      * *
*        * *        T                      * *
* INVOKER **                               * *
*********************************************
* AUSBIN   **        X                  X  * *
* AUSCRM   * *       X         X        X  * *
* AUSMLD   *  *      X                  X  * *
* AUSPRT   *   *     X                  X  * *
* BERBIN   *    *X X      X         X   XX * *
* BERTI    *     *                    X  * *
* BEWERT   *      *           X        X  * *
* BLIND    *       *                  X  * *
* BLOCKDAT *        *                    * *
* CNTRDR   *         *                   * *
* DATANF   *          *                X  * *
* EINBIN   *           *               X  * *
* EINMES   *            *              X  * *
* EINPAR   *         X   *             X  * *
* FEST     *        X     *   X        X  * *
* INIT     *   X      X  XX*X X X   X XX  * *
* KINIT    *                *           X  * *
* MAIN     *XXX X X    XX  X *XX XX  X X X* *
* MELDNG   *         X       *          X  * *
* PAREIN   *            X    X*X   X   X  * *
* PLAUS    *         X          *       X  * *
* PRENDE   *         X           *      X  * *
* PROTA    *   X                  *     X  * *
* PRUEF    *                  X    *    X  * *
* SKALL    *         X             *   X  * *
* SKALMw   *                       * X  * *
* STOP     *  X      X          X    *X  * *
* TRACE    *                          *  * *
* WANBIN   *                  X        X* *X*
* ZYKLUS.  *         X                X ** *
*********************************************
```

THE FOLLOWING MODULES ARE NOT INVOKED BY ANY MODULE ON THE LIBRARY

BLOCKDAT MAIN

THE FOLLOWING MODULES DO NOT INVOKE ANY MODULE ON THE LIBRARY

BLOCKDAT CNTRDR TRACE

Abb. 8: Matrix der Modulaufrufe

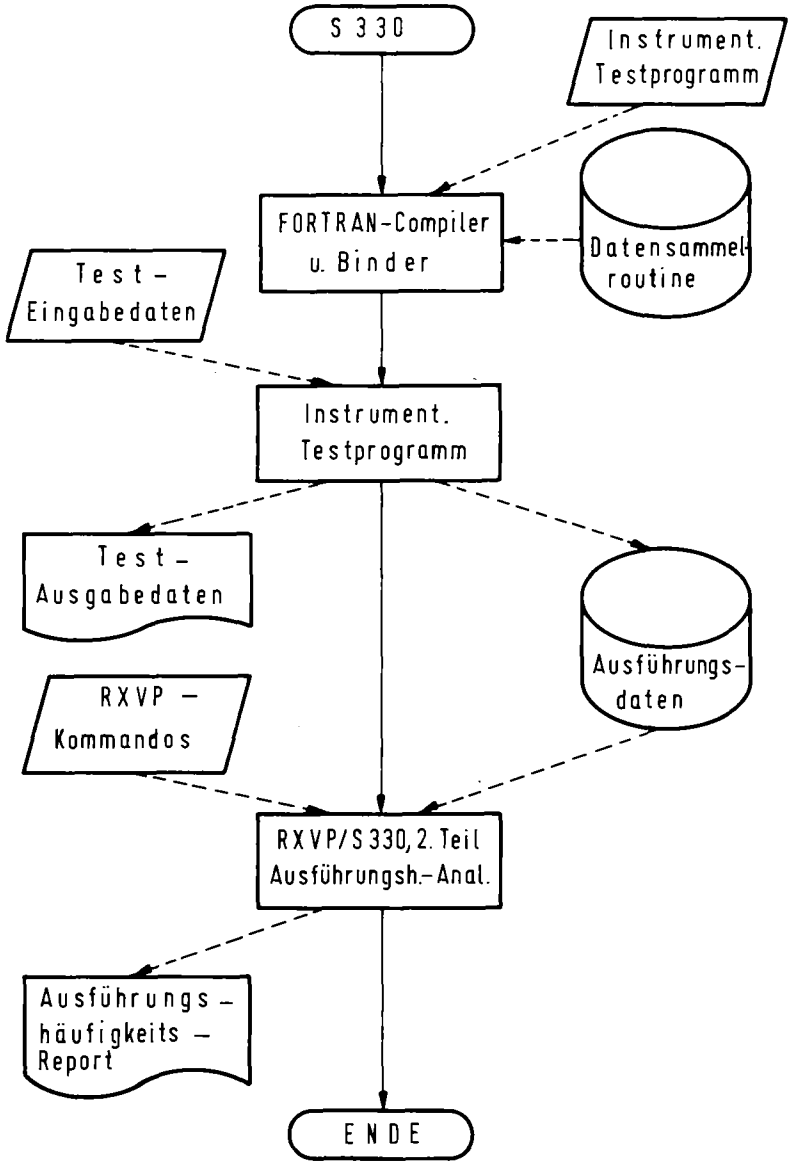


Abb. 9: Ablauf und Funktionen von RXVP/S330, 2. Teil
Ablauf auf der S330

Diese Daten werden anschließend vom 2. Teil von RXVP/S330 auf der S330 analysiert und es werden - wiederum gesteuert über Kommandos - verschiedene Ausführungshäufigkeits-Reports ausgegeben.

Für jeden gewünschten Modul wird ein detaillierter Ausführungshäufigkeits-Report ausgegeben (Abb. 10). In ihm wird für jede Verzweigungsalternative (die Nummern der Verzweigungsalternativen wurden u.a. in Abbildung 2 angegeben) die Anzahl der Ausführungen während des Testlaufs angegeben und das auf das Maximum normierte Ausführungsprofil graphisch dargestellt. Die nicht ausgeführtenVerzweigungen werden in der 2. Spalte extra gekennzeichnet.

Außerdem wird ein zusammenfassender Ausführungshäufigkeits-Report ausgegeben (Abb. 11), in dem für alle Module und für das gesamte Programmsystem die Anzahl der Aufrufe, die Anzahl der durchlaufenen Verzweigungen und der Prozentsatz der Programmüberdeckung durch die Testfälle angegeben wird. Schließlich werden noch die nicht durchlaufenen Verzweigungsalternativen gesondert aufgelistet (Abb. 12).

5. Aufwand beim Einsatz

Im letzten Kapitel wurden die wichtigsten Funktionen des Testsystems RXVP/S330 beschrieben, es wird nun noch auf den Aufwand eingegangen, der mit der Verwendung des Testsystems verbunden ist. Der Aufwand ist stark abhängig von dem konkreten Programm, welches untersucht wird. Es können deshalb nur typische Werte angegeben werden.

Die Kosten des 1. Teils von RXVP auf der IBM/370 für ein Testprogramm betragen etwa das 5- bis 15-fache der Übersetzungs- plus Bindekosten des Testprogramms.

Der Code des instrumentierten Programms auf der S330 wird gegenüber dem uninstrumentierten um etwa folgende Werte verlängert:

- varianter Teil (Datenteil) 0 %
- invarianter Teil (Befehlsteil) 20-80 %
- Datensammelroutine 4 kW

Die Laufzeit eines Testprogramms wird durch die Instrumentierung um ca. 50-200 % verlängert. Diese Daten beziehen sich auf eine Instrumentierung des gesamten Testprogrammsystems. Wenn nur Teile des Testprogrammsystems instrumentiert werden, werden auch der Code und die Laufzeit entsprechend geringer verlängert.

```
RECORD OF DECISION TO DECISION (DD PATH), EXECUTION

MODULE  $AUSPRT $          TEST CASE NO.        1

DD PATH I  NO.  NOT EXECUTED  I   NUMBER OF EXECUTIONS -- NORMALIZED TO MAXIMUM       I        I  NUMBER OF
NUMBER  I                     I.-------20.-------40.-------60.-------80.------100.    I        I  EXECUTIONS
        I                     I
    1   I                     I XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  I    1   I        9
    2   I                     I XXXXXXXXXXX                                          I    2   I        2
    3   I                     I XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX             I    3   I        7
    4   I                     I XXXXXXXXXXX                                          I    4   I        2
    5   I                     I XXXXXXXXXXXXXXXXXXXXXXXXXX                           I    5   I        5
    6   I                     I XXXXXXXXXXX                                          I    6   I        2
    7   I                     I XXXXXXXXXXXXXXXX                                     I    7   I        3
    8   I                     I XXXXX                                                I    8   I        1
    9   I                     I XXXXXXXXXXX                                          I    9   I        2
   10   I                     I XXXXX                                                I   10   I        1
   11   I                     I XXXXX                                                I   11   I        1
   12   I                     I XXXXX                                                I   12   I        1
   13   I  13          00000 I                                                      I        I

                                                          TOTAL NUMBER OF DD PATH EXECUTIONS =      36

  TOTAL OF   1  NOT EXECUTED     EXECUTED  12/ 13        PERCENT EXECUTED =    92.31
```

Abb. 10: Detaillierter Ausführungshäufigkeits-Report

```
                                  S U M M A R Y --  T H I S   T E S T

  TEST I  MODULE     NUMBER OF   I  NUMBER OF     D-D PATHS     PER CENT
  CASE I   NAME      D-D PATHS   I INVOCATIONS    TRAVERSED     COVERAGE
     1 I
       I  MAIN            3     I       1            3         100.00
       I  INIT           25     I       1           18          72.00
       I  TRACE           1     I     626            1         100.00
       I  DATANF          1     I       1            1         100.00
       I  EINPAR         11     I      11            9          81.82
       I  AUSPRT         13     I       9           12          92.31
       I  SKALL          19     I      11           18          94.74
       I  PLAUS          47     I      11           38          80.85
       I  EINMES          9     I      11            8          88.89
       I  SKALMW          7     I      11            7         100.00
       I  BERBIN          1     I      11            1         100.00
       I  BERTI           7     I      11            7         100.00
       I  BLIND           5     I      11            5         100.00
       I  FEST           11     I      11           11         100.00
       I  MELDNG          3     I     383            2          66.67
       I  CNTRDR          1     I     417            1         100.00
       I  PRUEF          31     I      11           30          96.77
       I  WANBIN         27     I      11           26          96.30
       I  AUSBIN          5     I      11            4          80.00
       I  EINBIN         13     I      22           12          92.31
       I  BEWERT         11     I      11           11         100.00
       I  AUSCRM          5     I      11            4          80.00
       I  AUSMLD          3     I      11            3         100.00
       I  PAREIN         23     I      11           18          78.26
       I  PROTA          27     I      11           27         100.00
       I  ZYKLUS          1     I      11            1         100.00
       I  PRENDE          1     I       1            1         100.00
       I                        I
       I  $$ALL$$       389     I                  277          89.64
       I                        I
```

Abb. 11: Zusammenfassender Ausführungshäufigkeits-Report

MODULE NAME	TEST NUMBER	PATHS NOT HIT	LIST OF DECISION TO DECISION PATHS NOT EXECUTED
<MAIN >	1	0	
	CUMUL	0	
<INIT >	1	7	10 14 16 18 20 22 24
	CUMUL	7	10 14 16 18 20 22 24
<TRACE >	1	0	
	CUMUL	0	
<DATANF >	1	0	
	CUMUL	0	
<EINPAR >	1	2	9 10
	CUMUL	2	9 10
<AUSPRT >	1	1	13
	CUMUL	1	13
<SKALL >	1	1	19
	CUMUL	1	19
<PLAUS >	1	9	8 16 20 32 35 36 40 44 46
	CUMUL	9	8 16 20 32 35 36 40 44 46
<EINMES >	1	1	4
	CUMUL	1	4
<SKALMW >	1	0	
	CUMUL	0	
<BERBIN >	1	0	
	CUMUL	0	

:

Abb. 12: Liste der nicht durchlaufenen Verzweigungsalternativen

Der zweite Teil von RXVP auf der S330 hat eine Programmlänge von 37 kW. Die Laufzeit dieses Teils des Testsystems für ein Testprogramm beträgt grob das 1- bis 5-fache bezogen auf die Ausführungszeit des uninstrumentierten Testprogramms.

6. Einsatzgebiete und Erfahrungen

Abschließend will ich noch auf die Einsatzgebiete von RXVP/S330 und die mit dem Testsystem gewonnenen Erfahrungen eingehen.

Das Testsystem RXVP wird seit Dezember 1977 auf der IBM/370 eingesetzt. Die S330-Version steht erst seit Februar 1978 zur Verfügung. Die S330-Version wurde eingesetzt u. a. beim Testen der Pilotimplementierung eines Schutzsystems /Gme 78/, bei der Entwicklung eines Datenbanksystems /Pol 78/ und bei der Installierung von FORTRAN-Programmsystemen auf der S330.

Das Testsystem hat sich dabei als sehr wertvolles Hilfsmittel erwiesen. Es hat sich gezeigt, daß es am sinnvollsten in der späten Implementierungsphase (wenn das Programmpaket eine leicht überschaubare Größe überschritten hat), in der Abnahmetestphase und in der Wartungsphase (inkl. Modifikationen) eingesetzt wird. Es wird primär verwendet, um das 'Ausgetestetsein' eines Programms, d.h. den Grad der Programmüberdeckung durch die Testfälle, zu prüfen und festzustellen, welche Programmteile noch nicht ausgetestet wurden, um schließlich eine vollständige Überdeckung zu erzielen. Damit kommt das Testsystem einem zentralen Problem im Abnahmetest entgegen. Weiter wird das Testsystem eingesetzt, um den Code eines Programmsystems zu dokumentieren, u.a. die Modulaufrufhierarchie und die Variablenverwendung. Diese Dokumentation hat sich insbesondere beim Umgang mit fremden Programmsystemen als sehr hilfreich erwiesen, z.B. bei der Erweiterung von Programmsystemen oder bei Installationen und den dabei notwendigen Änderungen. Bei einer Installation, bei der eine Reihe von Anpassungen vorgenommen werden mußten, wurde durch Verwendung des Testsystems der Installationsaufwand etwa halbiert.

Das Testsystem kann aber auch noch in weiteren Bereichen eingesetzt werden. Wenn z.B. die Laufzeit eines Programms reduzient werden muß, liefert es automatische Statistiken über die Ausführungshäufigkeit der einzelnen Module und innerhalb der Module der einzelnen Programmteile; es können dann gezielt die am häufigsten ausgeführten Programmteile zur Optimierung oder Implementierung in Assembler ausgewählt werden.

7. Weitere Arbeiten

Es ist geplant, das dargestellte Testsystem RXVP/S330 weiter auszubauen.

Das Testsystem soll so erweitert werden, daß es den gesamten Sprachumfang von PROZESS-FORTRAN 300 verarbeiten kann; hierzu muß im wesentlichen noch die Möglichkeit, FORTRAN und Assembler-Code zu mischen, eingearbeitet werden (siehe Kap. 3).

Weiter ist geplant, die Instrumentierung der Kontrollstruktur auf der S330 zu implementieren (siehe Kap. 4). Damit kann dann die gesamte dynamische Analyse (Ausführungshäufigkeits-Analyse) direkt auf der S330 durchgeführt werden.

Ferner sollen gewisse Prüfungen in das Testsystem eingebaut werden. Bisher wird weitgehend nur aufbereitete Information über das Testprogramm ausgegeben, die visuell auf Fehler abgeprüft werden muß. Zum Beispiel werden in Abbildung 3 die Unterprogramm-Definition und die Unterprogramm-Aufrufe ausgegeben; mit Hilfe dieser Abbildung kann visuell leicht auf Fehler in der Anzahl der formalen und aktuellen Parameter geprüft werden. Prüfungen dieser Art sollten aber besser automatisch durchgeführt werden.

L i t e r a t u r :

/Boe 73/ Boehm, B.W.
Software and Its Impact: A Quantitative Assessment
Datamation 19 (1973) 5, pp. 48-59

/Boe 76/ Boehm, B.W.; Brown, J.R.; Lipow, M.
Quantitative Evaluation of Software Quality
Proc. of the 2nd Intern. Conf. on Software Engineering,
San Francisco, 1976, pp. 592-605

/Els 76/ Elshoff, J.L.
An Analysis of Some Commercial PL/I Programs
IEEE Trans. on Software Engineering, 2 (1976) 2, pp. 113-120

/Gen 77a/ General Research Corporation
RXVP Fortran Automated Verification System,
User's Guide
Santa Barbara, April 1977

/Gen 77b/ General Research Corporation
RXVP Fortran Automated Verification System,
Reference Manual,
Santa Barbara, April 1977

/Gme 78/ Gmeiner, L.; Lemperle, W.; Voges, U.
Mehrfachimplementierung eines Reaktorschutzsystems in PHI2,
PASCAL und IFTRAN auf der S330,
9. Jahrestagung des Siemens-Prozeßrechner-Anwenderkreises I
Karlsruhe, April 1978, KfK 2642

/Hua 75/ Huang, J.C.
An Approach to Program Testing
ACM Computing Surveys 7 (1975) 3, pp. 113-128

/Mil 77/ Miller, E.F.
Program Testing: Art Meets Theory
Computer 10 (1977) 7, pp. 42-51

/Pol 78/ Polster, F.-J.
FADABS: Ein Datenbanksystem für den Siemens Prozeßrechner 330
9. Jahrestagung des Siemens-Prozeßrechner-Anwenderkreises I
Karlsruhe, April 1978, KfK 2642, S. 227-244

H. Buseck
Bundesanstalt für Straßenwesen
Köln

Einsatz einer S 310 für Analogdatenverarbeitung

1. Einführung

Bei der Bundesanstalt für Straßenwesen in Köln - einer dem Bundesverkehrsministerium nachgeordneten technisch-wissenschaftlichen Behörde - fallen eine ganze Reihe meßtechnischer Probleme an, deren Lösung in zunehmendem Maße auch mit Hilfe von Prozeßrechnern vorgenommen wird.

Viele von der Bundesanstalt eingesetzte Routinemeßverfahren wurden so umgestaltet, daß sie ihre Meßergebnisse auf Datenträgern aufzeichnen, 5- oder 8-Kanal-Lochstreifen, Analogmagnetbänder, Magnetkassetten o.ä. Die große Datenverarbeitungsanlage ist nicht imstande, diese Datenträger unmittelbar zu lesen, sie versteht nur Computerband. Auch an dieser Stelle werden Prozeßrechner eingesetzt.

Es soll hier keine hochwissenschaftliche oder technisch komplizierte Abhandlung gebracht werden, ich will nur ein Beispiel geben für das, wonach dieser Kreis benannt ist, für die Anwendung von Siemens-Prozeß-Rechnern zur Lösung von Aufgaben.

Auch bei der einfachen Verarbeitung von Analog- und Digitaldaten können Probleme auftreten, die zwar auf einer niedrigen technischen Ebene liegen, aber ärgerlich genug sein können.

Aus diesen Gründen und um den Einsatz von Siemens Prozeßrechnern bei der Bundesanstalt zur allgemeinen Information zu schildern, wird über den "Einsatz einer S 310 für Analogdatenverarbeitung" berichtet.

Dies geschieht am Beispiel einer uns gestellten Aufgabe, wobei der Aufgabensteller selbst die Möglichkeit eines Prozeßrechners nicht kannte.

2. Die Aufgabe

Die Bundesanstalt betreibt im Rahmen eines über längere Zeit laufenden Meßprogramms eine Reihe sogenannter "Achslastwaagen". Das sind Meßeinrichtungen, die die Achsgewichte von Fahrzeugen erfassen, ohne den laufenden Verkehr zu beeinträchtigen. Normalerweise wird gewogen, klassiert und in Form von Klassenhäufigkeiten stundenweise gespeichert.

Dazu bedienen wir uns verschiedener Verfahren, die wesentlichsten sind (siehe Bild 1):

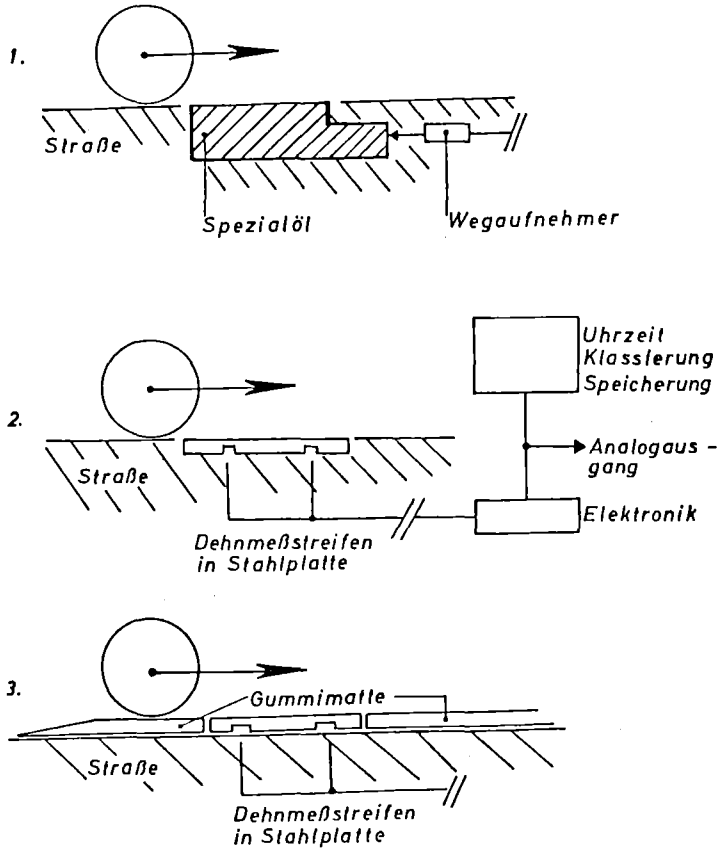


Bild 1
Drei Achslastwiegeverfahren

1. hydraulisches System, ortsfest
2. DMS-System, ortsfest,
3. DMS-System, geweglich.

Jedes Meßsystem hat Fehler oder Ungenauigkeiten und nur wenn diese bekannt sind, sind die Ergebnisse wertbar und verwertbar.

Zur Überprüfung dieser Meßsysteme wird ab und zu eine gemeinsame Wiegeaktion mit allen Systemen und allen beteiligten Abteilungen durchgeführt, bei der von allen Systemen der gleiche Verkehr erfaßt wird, d.h. daß exakt die gleichen Fahrzeuge gewogen werden. Dazu werden die Analog-Wiegesignale auf Band genommen und die amtlichen Kennzeichen dazu aufgesprochen. Bei der Auswertung wird das Ergebnis mit den von den Systemen selbst gespeicherten Ergebnissen verglichen.

Jetzt bin ich fast schon beim Thema, beim Prozeßrechner, denn jetzt beginnt die Auswertung.

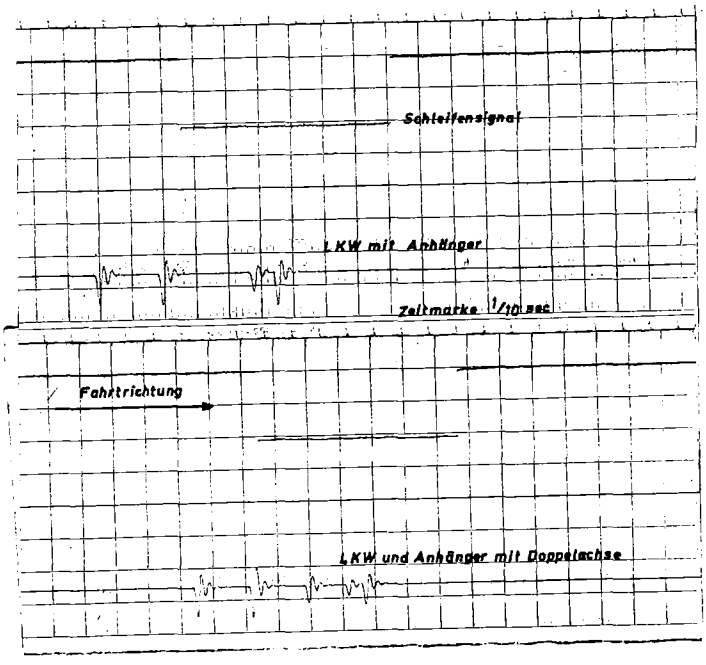


Bild 2
Kanal 1 (Wiegesignal) u. Kanal 3 (Schleifensignal) des Analogmagnetbandgerätes
Papiergeschwindigkeit: 8,6 [cm/sec]
Zeitmaßstab 1 [cm] ≙ 116 [ms]

Die bei den Überfahrten der Fahrzeuge - als Beispiel für einen Fall, das hydraulische, ortsfeste System - aufgezeichneten Analogsignale zeigen die Bilder 2 und 3. Es sind Schreiberaufzeichnungen mit unterschiedlichen Schreibgeschwindigkeiten.

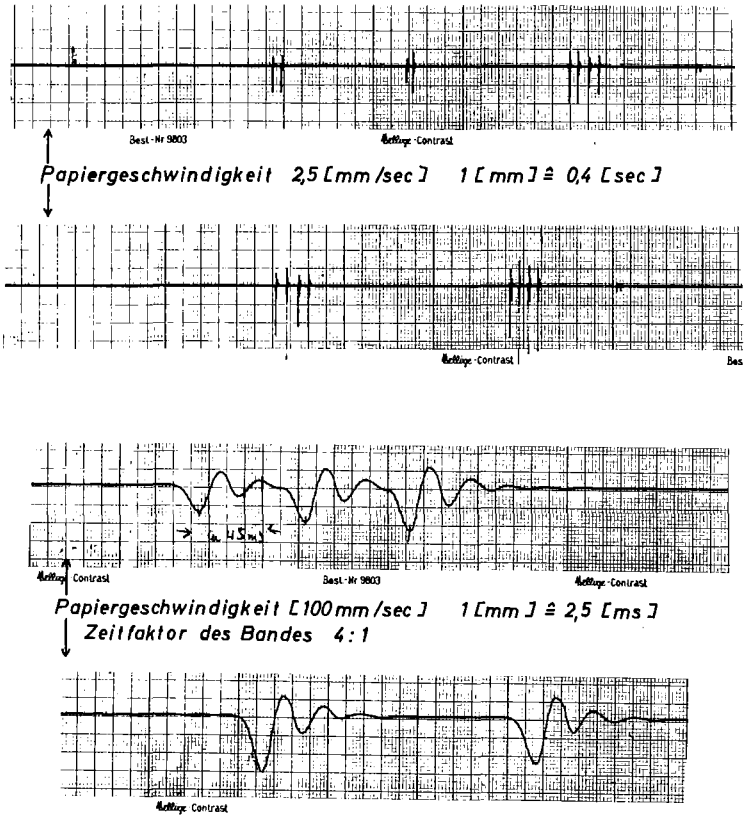


Bild 3
Kanal 1 (Wiegesignal) des Analogmagnetbandgerätes

Die Aufgabe bestand darin, die Größe der Achslast aller während der Meßzeit die Waage passierenden Lkw's zu ermitteln, wobei die zu einem Fahrzeug gehörigen Achsen zu kennzeichnen waren. Die auf dem Kommentarkanal festgehaltenen amtlichen Kennzeichen werden nach der Auswertung von Hand in die vom Prozeßrechner erstellte Liste eingetragen.

Als "Kalibriernormal" für die Bandaufzeichnung galt ein wiederholt im Verkehrsstrom mitfahrendes, exakt gewogenes Dienstfahrzeug einer Autobahnmeisterei.

Es wurde ein absoluter Fehler in der Lastangabe von ± 50 kg (± 0,5 kN) zugelassen.

Die Analogdaten waren auf einem FM-4-Spur-Analogmagnetbandgerät aufgezeichnet. Dessen Ausgangsspannung war ± 5 V. Bei Signalen, die nur im positiven oder negativen Spannungsbereich liegen, besteht die Möglichkeit, den Nullpunkt bei der Aufnahme zu verschieben, um so den Eingangsspannungsbereich besser ausnutzen zu können. Von dieser Möglichkeit wurde Gebrauch gemacht. Die Bandgeschwindigkeit bei der Aufnahme war 3 3/4 ips, d.h. etwa 9,5 cm/sec.

3. Die Lösung

Es stand nicht von vornherein fest, daß zur Auswertung der Bänder der Prozeßrechner benutzt werden mußte. Dies ergab sich erst bei den anhand der Analogaufzeichnungen angestellten Überlegungen über das zweckmäßigste Auswerteverfahren.

Zwei Kriterien spielten bei dem zu wählenden Auswerteverfahren eine Rolle:

1. der Zeit- und Personalaufwand
2. die erreichbare Genauigkeit

Jedes für sich allein sprach schon für den Einsatz der 310.

Es wurde eine normale Zentraleinheit eingesetzt, mit Analog- und Digitaleingaben, mit Relaisausgabe, 24 k

Kernspeicherausbau und Lochstreifenein- und -ausgabe. Wir haben das ORG 310 und den AS 10 benutzt. Hier stecken keinerlei Besonderheiten.

Insgesamt lagen zur Auswertung 14 1/4-Zoll-Bänder vor, deren jedes 1 800 Fuß, entsprechend etwa 548 m lang war. Bei der Bandgeschwindigkeit von 3 3/4 ips = 9,5 cm/sec beträgt die Gesamtlaufzeit eines Bandes 5 760 sec = = 96 min = 1 Std. und 36 min.

Eine Auswertung dieser Bänder durch Aufzeichnen der Signale auf einem Direktschreiber und Ausmessen der Abstände vom Signalmaximum zur Null-Linie ist nahezu unmöglich, wie die folgenden Überlegungen zeigen sollen.

Als erstes wird ein Direktschreiber wie in Bild 2 bzw. 3 betrachtet. Um eine zur Auswertung ausreichende zeitliche Auflösung zu erhalten, ist eine Papiervorschubgeschwindigkeit von 100 mm/sec erforderlich. Dies ergibt bei einer Laufzeit von 5 760 sec je Band bei 14 Bändern einen - theoretischen - Papierstreifen von etwa 7 km Länge. Für alle 14 Bänder ergibt sich eine theoretische Schreiberlaufzeit von 1 344 min = 22,4 Std. Die Schreibbreite des Schreibers beträgt ± 20 mm, ein Signal habe im Mittel etwa 30 mm Höhe. Bei der Abstandsermittlung eines Maximuns von der Null-Linie muß mit einer Unsicherheit von ± 0,5 mm gerechnet werden. Dies führt zu einem relativen Fehler von ±1,6 % oder - eine 10 Mg-Achse vorausgesetzt - zu einem absoluten Fehler von ± 160 Kg (1,6 kN). Abschätzungen des Zeitbedarfs zur Auswertung sind müßig.

Wird ein Gerät mit einer ausreichenden Schreibbreite (30 cm) verwandt, muß wegen des geringen Frequenzumfanges solcher Geräte eine Zeittransformation von etwa 8 : 1 vorgesehen werden. Die ist nur unter Zuhilfenahme eines zweiten

Bandgerätes möglich. Damit ergibt sich eine Laufzeit bei der Wiedergabe von 12 Stunden je Band. Für alle 14 Bänder errechnet sich ein theoretischer Papierverbrauch von 1 935.36 m² und eine Schreiberlaufzeit von 168 Std - - 21 Arbeitstage, ohne das Überspielen auf ein zweites Band.

Eine dritte Möglichkeit, einen Speicherkathodenstrahloszillografen zu verwenden und nur die Signale, nicht die Zwischenzeiten zwischen zwei Achsen bzw. zwischen zwei Fahrzeugen aufzuzeichnen, entfällt, weil die verwendeten Bandgeräte keinen schnellen Start/Stop-Betrieb zulassen, der an sich über das Triggersignal des Oszillografen möglich wäre. Dabei reichte ebenfalls die erzielbare Genauigkeit für die Lasthöhenangabe nicht aus. Einen UV-Lichtstrahloszillografen habe ich in die Betrachtung jetzt nicht einbezogen.

Die einzige, im Rahmen der vorhandenen technischen Möglichkeiten realisierbare Lösung bestand im Einsatz des Prozeßrechners. Hier ging es darum, aufzuzeigen, für welche Fälle ein Prozeßrechner auch verwendet werden kann bzw. daß es Fälle gibt, wo eine andere Lösung praktisch nicht machbar ist. Dazu war diese Rechnerei mit Zeit- und Papierbedarf gedacht.

Es wurde ein kleines Programm geschrieben, ca. 1/2 K - siehe Flußdiagramm (Bild 4) - mit einigen Fehlerausgängen, teils mit, teils ohne Eingreifmöglichkeit für das Bedienpersonal. Einige Punkte möchte ich erwähnen:

1. Einschaltung eines Filters zur Rauschunterdrückung zwischen Band und Analogeingang.
2. Ständige Bildung eines Nullpunkt-Mittelwertes (8 Digitalwerte) - aber ohne Fahrzeuge.

3. Unterdrücken von Pkw-Achsen (Schwellwert)
4. Übersteuerungsausgang mit Bandstop über Relaisausgabe.
5. Jedes Fahrzeug hat maximal 7 Achsen
6. Trennung zwischen 2 Fahrzeugen durch 500 ms-Zeitschwelle.
7. Unterdrücken der Nachschwinger durch 30 ms-Totzeit.
8. Uhrzeitausgabe bei jeder 1. Achse, Uhrzeit ist Versuchszeit.
9. Achsabstand in 20 ms-Zeiteinheiten (PI-Baugruppe).
10. Fehlerausgang "Speicher voll" (16 K Worte).
11. Fehlerwarteschleife mit Reaktion des Bedieners.

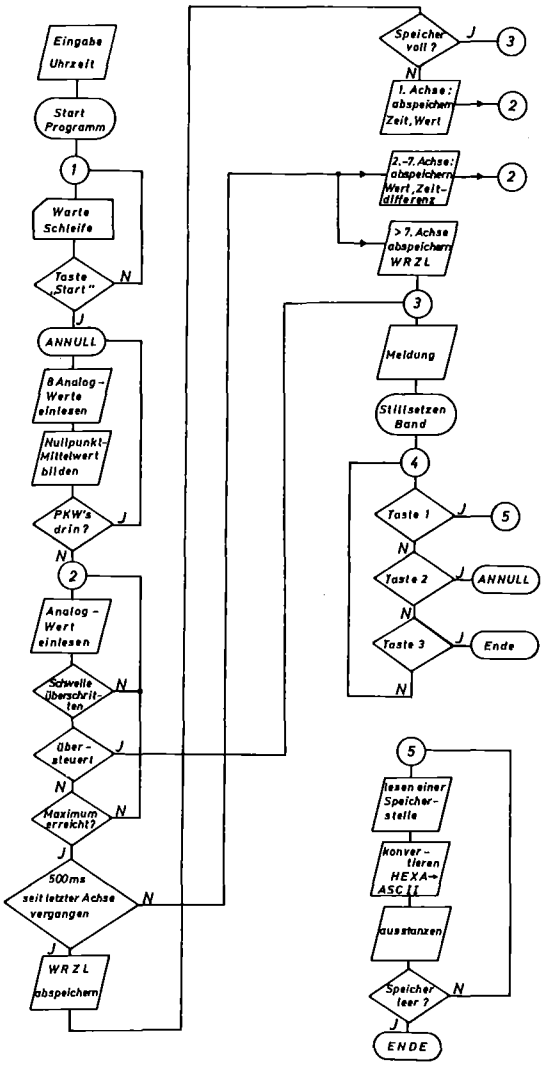


Bild 4
Flußdiagramm

Zum Schluß erfolgt die Datenausgabe (einmal je Band) über die Lochstreifenausgabe, die leider etwas langsam ist. Ein Beispiel für diese Datenausgabe zeigt Bild 5.

H	M	S	A1	Z1	A2	Z2	A3	Z3	A4	Z4	A5
Z5	A6	Z6	A7								
15	15	27	1303	280	1669						
15	15	33	822	200	1315						
15	16	20	773	180	1023						
15	16	39	1293	360	1133	300	497	260	756		
15	16	42	1112	240	1644						
15	16	46	1441	280	2091	320	1654	340	1547		
15	16	57	1333	280	1263	260	633	340	430		
15	17	5	1424	340	1502	460	894	130	893		
15	17	11	1627	240	2214	260	1514	100	1714	80	1870
15	17	13	480								
15	17	17	1154	340	1571	400	984	300	774		
15	17	19	539								
15	17	39	801	200	684						
15	17	46	1810	220	1908	60	1354	240	1309	260	1784
15	18	6	1505	200	1986	320	1745	40	1808	100	1554
15	18	48	762	260	669	300	563				
15	18	51	1317	320	1313						
15	19	12	514								
15	19	44	1258	200	778	240	476	30	622		
15	19	53	790								
15	20	1	1232	260	958	260	853	260	618	80	605
15	20	6	1116	300	1026	340	685	260	564		
15	20	20	1526	180	2111	220	1677	80	1519	60	1624
15	20	38	1720	200	1958	80	1847	180	1217	100	1120
80	866										
15	20	40	504								
15	21	21	572	200	660	460	609				

Bild 5 Ergebnisausdruck über Bedienungsblattschreiber

Die Werte sind so nicht verwendbar, zwischen dem Analog-Wert (in Einheiten) und der Achslast steckt erstens ein Faktor, der z.B. das Übersetzungsverhältnis des Analogbandes und den Meßbereich des ADW enthält, zum zweiten die nichtlineare Eichkurve des Wiegesystems.

Diese wurde in einer Liste gespeichert und über die große Rechenanlage umgerechnet, weil das in der 310 zu zeitunrentabel wäre.

Die Bedeutung der Zahlen (bzw. der Symbole der Überschriftzeile in Bild 5 ist folgende:

Über-schrift-Zeile	1.Zahlen-zeile in Bild 5	Bedeutung
H	15	Stundenwert bei der Messung = 15.00 Uhr
M	15	Minutenwert bei der Messung = 15.30 Uhr
S	27	Sekundenwert beim Übergang der 1. Achse eines Fahrzeuges
A_1	1 303	Zahlenwert für die 1. Achse
Z_1	280	Zeitabstand zwischen 1. und 2. Achse
A_2	1 669	Zahlenwert für die 2. Achse

usw.

Anhand der Zeitangaben (H,M,S) läßt sich der Zeitpunkt der Überfahrt der ersten Achse eines Fahrzeuges über die Waage exakt feststellen, wenn die Zeituhr des Prozeßrechners zu Beginn des Durchlaufs eines Analogbandes auf die auf dem Band aufgesprochene Anfangszeit eingestellt wird. So ist eine bessere Zuordnung zwischen der einem Fahrzeug entsprechenden Zeile und der Angabe des Kennzeichens auf dem Kommentarkanal gegeben. Aus der Sekundenangabe (S) läßt sich der Fahrzeugabstand ermitteln - eine weitere Hilfe bei der Zuordnung.

Der Zeitabstand zwischen den Achsen (Z_1 bis Z_6, mehr als 7 Achsen sind im Programm nicht vorgesehen) erlaubt eine grobe Klassifikation der Fahrzeugart (Lkw, Lkw mit Anhänger, Sattelschlepper usw.), wenn eine einheitliche Fahrgeschwindigkeit angenommen werden kann. So kennzeichnen z.B. Zeiten von 40 bis 80 ms Doppelachsen von Fahrzeugen oder Anhängern.

Ein Fahrzeug - jeweils eine Zeile im Ausdruck - gilt dann als vorbeigefahren,wenn die nächste Achse mehr als 500 ms später als die voraufgegangene die Waage passiert. Danach erwartet das Programm ein neues Fahrzeug.

4. Die Kalibrierung

Die Kalibrierung umfaßt im vorliegenden Fall nur die Beziehung zwischen der Signalspannungsgröße auf dem Magnetband und dem vom Rechner gespeicherten bzw. ausgegebenen Zahlenwert (A_1 bis A_7). Das Verhältnis zwischen Eingangs- und Ausgangsspannungswert des Magnetbandes läßt sich mit hoher Genauigkeit feststellen und war ebenso wie die Eichkurve der Waage in das erwähnte FORTRAN-Programm eingearbeitet.

Zur Durchführung dieser Kalibrierung wurden vom Magnetband willkürlich einige Achssignale ausgewählt und jeweils eins in einen Digital-Speicher eingeschrieben (Siemens).

Dieser Speicher hat eine Auflösung der Zeitachse in 2048 Zeitschritte, eine Auflösung des Meßsignals in 256 Schritte (8 bit), was einer Auflösung auf $\pm$ 0,25 % entspricht. Das Signal wird in eine Halbleitermatrix eingeschrieben und kann wiederholt ausgegeben werden, repetierend auf einem Kathodenstrahloszillografen oder zur Eingabe in den Prozeßrechner, oder jeweils einmal auf einem X-Y-Schreiber. Aufnahme- und Wiedergabezeit sind getrennt wählbar im Bereich von 10 ms bis 500 sec.

Als Aufnahmezeit wurde 400 ms gewählt. Bei repetierender Wiedergabe mit der gleichen Zeitbasis erscheint daher alle 400 ms ein Achssignal (immer das gleiche).

Zwischen das Bandgerät und den Speicher wurde ein Tiefpaßfilter mit einer oberen Grenzfrequenz von 80 Hz zur Rauschunterdrückung geschaltet. Bild 6 zeigt die Wiedergabe dieses Achssignals auf einem X-Y-Schreiber. Die Zeitbasis (400 ms) ist markiert, 1 mm auf dem Papier entspricht einer Zeit von 1.1 ms. Die Auswertung

des Achssignals durch den Prozeßrechner wurde 128 mal wiederholt. Es traten Abweichungen im Zahlenwert für Achssignal nur im Rahmen des Digitalisierungsfehlers auf (± 1 Einheit), der Achsabstand betrug 400 ms (Zeitbasis des Speichers).

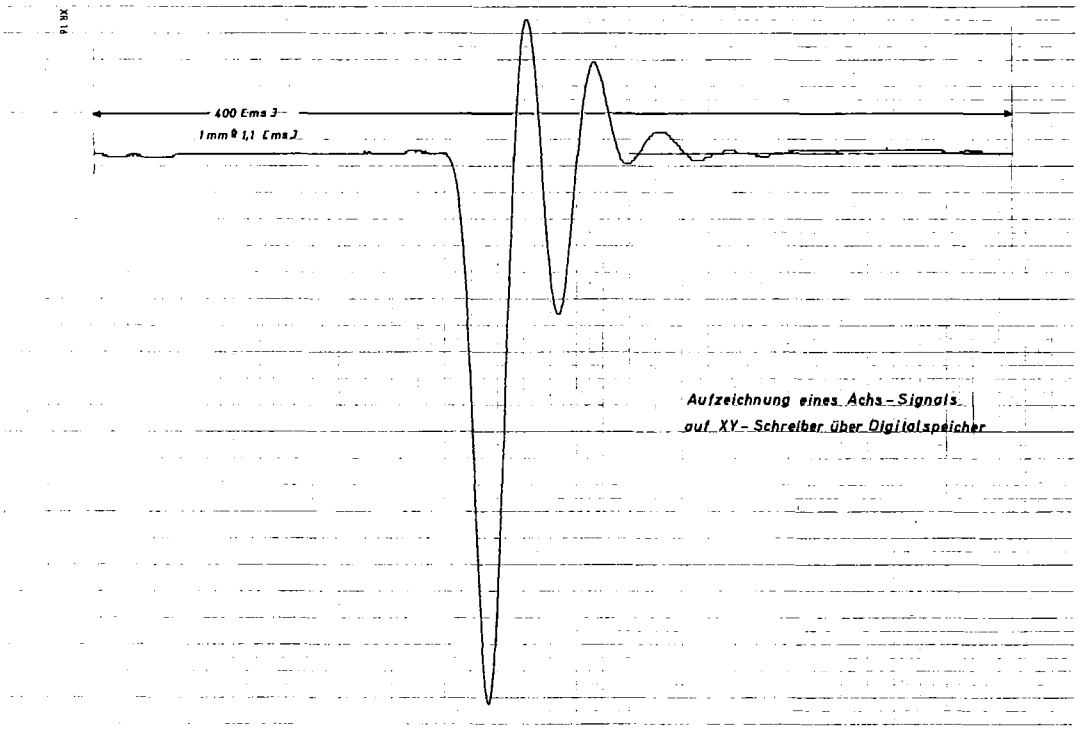


Bild 6 Aufzeichnung eines Achssignals

Jeweils für 7 Signale wurde ein neuer Nullpunkt berechnet. Dies geschieht durch Mittelbildung aus 8 im Zeitabstand von 1 ms eingelesenen Werten. Hierbei tritt naturgemäß eine Streuung auf, da die Null-Linie schwankt und der Zeitpunkt der Nullpunkt-Berechnung nicht immer der gleiche ist. Diese Nullpunktschwankung hängt mit den Eigenschaften des Analogmagnetbandgerätes zusammen und ist nicht zu umgehen. Aus den 26 Nullwerten ergibt sich ein Mittelwert für den Achs-Zahlenwert von

A = 1216.4 mit einer Standardabweichung von
s = 9.2

Daraus errechnet sich ein mittlerer Fehler von ± 0.75 %. Zur Kontrolle des Achs-Zahlenwertes selbst wurde das Achssignal aus dem Speicher so zeitgedehnt auf dem X-Y-Schreiber ausgegeben, daß 1 mm auf dem Papier einer Zeit von 160 µs entspricht (Bild 7). Dort kann der Abstand Null-Linie/Maximun ausgemessen werden (206 mm). Mit einem Spannungsnormal war der Schreiber vorher kalibriert worden. Es ergab sich dabei eine vertikale Auslenkung von 135.5 mm für eine Spannungsdifferenz von 4 000 mV. Dies entspricht einer Empfindlichkeit von 29.52 mV/mm.

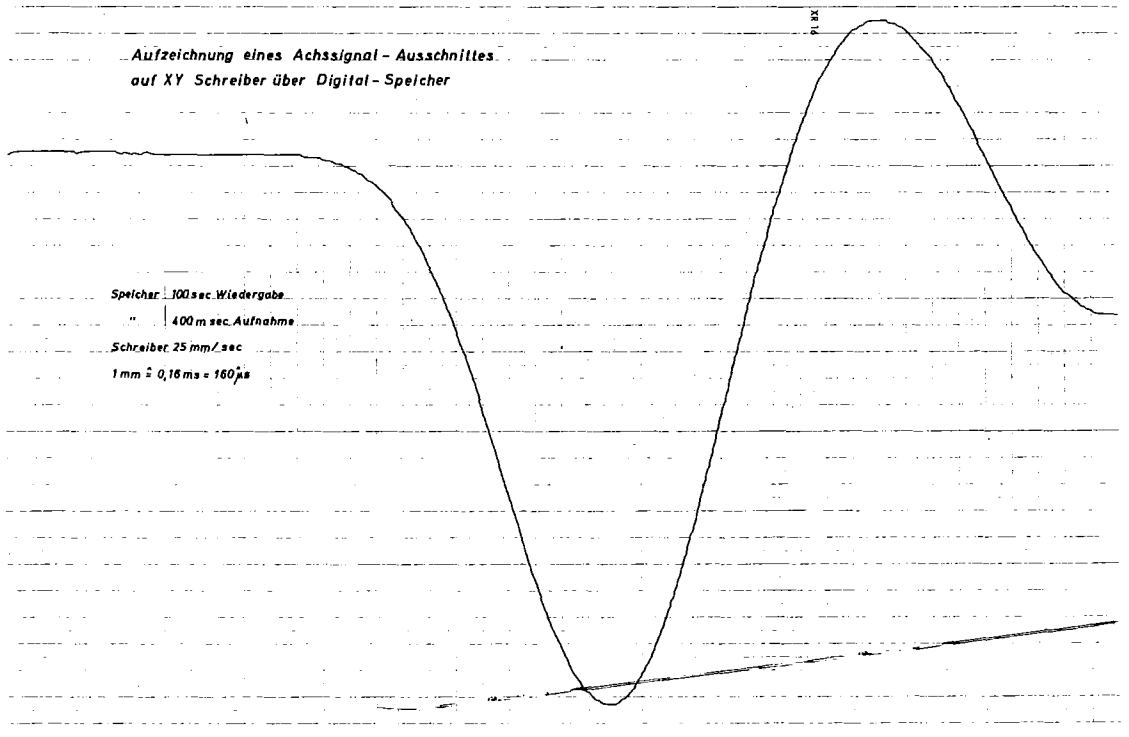


Bild 7 Stark zeitgedehntes Achssignal

Damit beträgt die Spannung des wiederholt ausgegebenen und vom Prozeßrechner ausgewerteten Achssignals 6.084 V.

Im Prozeßrechner wurde ein Meßbereich von 10 V gewählt. Dieser Spannungswert wird in 2047 Schritte zerlegt

(Auflösung ± 0,5 %). Ein Achs-Zahlenwert von 1219.4 entspricht somit einer Spannung von 5.957 V, d.h. eine um 2 % zu geringe Anzeige des Prozeßrechners gegenüber dem Eichnormal.

Bei der Auswertung des wiederholt ausgegebenen Digitalspeicher-Signals wurde jedoch zwischen Speicher und Rechner ein Filter geschaltet, um die Störungen durch das Bandrauschen zu unterdrücken. Dieses Filter konnte bei der langsamen Ausgabe auf dem X-Y-Schreiber nicht verwendet werden, war also nicht eingeschaltet. Für das Filter ergab sich im Mittel ein Übertragungsverhältnis von 0,98. Wird demgemäß die auf dem X-Y-Schreiber ermittelte Spannung von 6.048 V mit dem Übertragungsfaktor des Filters von (gerundet) 0.98 multipliziert, so erhielt der Rechner ein Spannungssignal der Höhe 5.927 V, das mit einer Abweichung von 0.5 % dem mittleren Meßwert des Rechners entspricht.

Der Auswertevorgang läuft in "Echtzeit" ab, d.h. es wird bei der Wiedergabe (Auswertung) die gleiche Bandgeschwindigkeit benutzt, wie bei der Aufnahme. Während der Auswertung kann daher der gesprochene Kommentar mitgehört werden, was bei Zeittransformation über das Bandgerät nicht möglich wäre.

5. Einige Schlußbemerkungen

Bei der Umsetzung eines Analogwertes in einen digitalen braucht der Analog-Digital-Wandler des Prozeßrechners eine Zeit von 132 µs. Für die weitere Verarbeitung im Programm sind etwa 30 Ein-Wort-Befehle notwendig. Dies ergibt nochmals 108 µs, ehe eine neue Umwandlung beginnt. Alls 240 µs wird also die Signalspannung abgetastet und der jeweils neue Wert mit dem vorhergehenden verglichen. Ist er kleiner, gilt der vorhergehende als Extremwert. Das funktioniert gut, solange der Anstieg der auszuwertenden Kurve ausreicht. Wird der Anstieg zu flach, so daß sich der digitalisierte Wert nur um

ein oder zwei Einheiten zwischen zwei Abtastungen ändert, so gerät er in die Größe der Digitalisierungsfehler - bei Differenzbildung ± 2 Digit - und die Sache klappt nicht mehr. Wird zur Abhilfe die Zeit zwischen zwei Abtastungen vergrößert, dann sinkt die Genauigkeit, mit der der Extremwert erkannt wird. Zur gleitenden Mittelwertbildung fehlt - vor allem wenn mehrere Analogkanäle abgefragt werden müssen - oft die Zeit. Das bedeutet: Zeitlich veränderliche Vorgänge mit sich stark änderndem Differentialquotienten machen Schwierigkeiten bei der Digitalisierung. Bei einem anderen Problem haben wir uns so geholfen, daß wir den Differentialquotienten abgefragt und mit ihm die Zeit zwischen zwei Digitalisierungen gesteuert haben.

Auf die hier beschriebene Art und Weise lassen sich selbstverständlich auch andere, das Verhalten einer Straßenbefestigung beschriebende mechanische Größen - wie Druck, Dehnung, Temperatur usw. - messen, klassieren und speichern. Ein solches Meßprogramm mit Verwaltung, Fehlererkennung- und Reaktion ist in Arbeit, und wird Ende des Jahres einsatzbereit sein. Die Endverarbeitung der Meßwerte wie Statistik, Korrelation usw. erfolgt dann in der Groß-Datenverarbeitungsanlage.

Ein Interface für den Blockverkehr über die EA-Schnittstelle des Siemens-Prozeßrechners 330

H.-J. Schuster

Physikalisch-Technische Bundesanstalt, Braunschweig

Über die EA-Schnittstelle wird der Datenaustausch zwischen einem Siemens-Prozeßrechner 330 und Prozeßgeräten abgewickelt. Er kann zentral oder peripher initiiert, d.h. vom Rechner bzw. von den Peripheriegeräten angeregt werden.

Der zentral initiierte Datenaustausch ist bezüglich des Interface relativ leicht zu realisieren. Er kann aber in der Praxis oft nicht angewendet werden, weil die Zeiten, zu denen ein Datenaustausch stattfinden soll, von den Peripheriegeräten bestimmt werden müssen.

Beim peripher initiierten Datenaustausch gibt es prinzipiell zwei Möglichkeiten:

1. Durch einen Organisationsalarm wird ein Programm geweckt, das ein Datenwort zentral initiiert überträgt und dann auf den nächsten Alarm wartet.

2. Die Daten werden durch Datenalarme ohne Mitwirkung eines Programmes direkt in den Arbeitsspeicher (DMA-Mode) übertragen.

Der erste Fall ist einfach zu realisieren, hat aber den Nachteil, daß die Reaktionszeit auf einen Organisationsalarm

sehr groß ist - ca. 500 µs -, wenn einmal von der Schnellreaktionsebene des Rechners abgesehen wird, so daß auf diese Weise nur etwas mehr als 1000 Worte/s übertragen werden können. Diese Summenrate ist aber in der Regel - besonders wenn wie in der Physikalisch-Technischen Bundesanstalt (PTB) mehrere Prozesse an einem Rechner hängen - viel zu gering.

Der zweite Fall des peripher initiierten Datenaustauschs ist sehr viel schneller. Es können bis zu ca. 300 000 Worte/s übertragen werden. Dieser Blockverkehr ist zwar bei der EA-Schnittstelle vorgesehen, aber leider in der Prozeßeinheit 3600 nicht vollständig implementiert. Es fehlen Prozeßsignalformer und einige Steuerfunktionen in der Grundsteuerung 3601. Daher mußte in der PTB ein Interface zur Implementierung des Blockverkehrs über die EA-Schnittstelle selbst geschaffen werden. Dieses Interface, das hier vorgestellt wird, ist ein Bestandteil der Ein-/Ausgabesteuerung, die in der PTB als Alternative zur Grundsteuerung entwickelt wurde und über die auf der letzten Jahrestagung vorgetragen worden ist. Die wesentlichen Gründe für die Entwicklung dieser Prozeßanschlußeinheit waren: Kosteneinsparung, Vereinfachung des Anschlusses nicht standardisierter Geräte und - worüber hier berichtet wird - die Implementierung eines leistungsfähigen Blockverkehrs.

Die funktionellen Anforderungen an das Interface zum Blockverkehr werden im wesentlichen von den Eigenschaften der EA-Schnittstelle bestimmt und sollen im folgenden hergeleitet werden. Die EA-Schnittstelle besteht hauptsächlich aus einem 16 bit-Dateneingang, einem 16 bit-Datenausgang und einem Alarmeingang für Organisations- und Datenalarme.

Durch Organisationsalarme werden darauf wartende Programme geweckt, durch Datenalarme Daten ohne Mithilfe eines Programms in den oder aus dem Arbeitsspeicher geschrieben. Mit einem Organisationsalarm ist die Alarmnummer oder Prozeßsignalformeradresse und mit einem Dateneingabealarm das Datenwort als Alarmbegleitinformation auf den Dateneingang der EA-Schnittstelle zu schalten. Vor der Bearbeitung von Datenalarmen ist zunächst dem EA-Prozessor, der die EA-Schnittstelle steuert, per Programm der Datenbereich im Arbeitsspeicher mitzuteilen, in den z.B. bei der Dateneingabe ein Datenblock eingeschrieben werden soll. Die mit den Datenalarmen angebotenen Daten werden dann vom EA-Prozessor selbständig im DMA-Mode fortschreitend in die Zellen dieses Datenbereiches geschrieben. Der EA-Prozessor gibt auch eine Meldung an das Interface, wenn der Datenbereich vollgeschrieben wurde und der EA-Prozessor neu versorgt werden muß.

Auf Grund dieser Eigenschaften der EA-Schnittstelle ergeben sich nun folgende Anforderungen an das Interface für den Blockverkehr:

1. Mit dem Datenalarm muß gleichzeitig das zu übertragende Datenwort an den Dateneingang angelegt werden.

2. Da der Dateneingang der EA-Schnittstelle gleichermaßen für zentral initiierte Eingaben und für die Alarmbegleitinformation der Alarme benutzt wird, muß ein Datenalarm gesperrt werden, wenn eine zentrale Initiative oder ein anderer Alarm bearbeitet wird. Sonst werden die Informationen am Dateneingang überlagert.

3. Weiterhin müssen Datenalarme gesperrt werden, wenn dem EA-Prozessor kein definierter Datenbereich zugewiesen oder dieser bereits voll geschrieben wurde. Sonst können undefiniert Zellen im Arbeitsspeicher überschrieben werden.

4. Zur Optimierung der Geschwindigkeit sollen Datenalarme, wenn keine Sperrbedingung entsprechend 2. und 3. vorliegt, unverzüglich zur EA-Schnittstelle durchgeschaltet werden.

5. Das Interface hat auch dafür zu sorgen, daß eine erfolgte Alarmbearbeitung beim alarmstellenden Gerät quittiert wird.

6. Der Blockverkehr soll ferner gleichzeitig über jede Anschlußstelle aller an eine EA-Schnittstelle angeschlossenen EA-Steuerungen möglich sein. Es können praktisch beliebig viele EA-Steuerungen an eine EA-Schnittstelle durch Kettung angeschlossen werden.

7. Schließlich soll das Interface auch geschützt sein gegen Blockierungen, wenn ein alarmstellendes Gerät im Fehlerfall einen bereits bearbeiteten Alarm nicht zurücknimmt.

Alle die genannten Anforderungen an ein leistungsfähiges Interface für den Blockverkehr können mit Hilfe einer sehr einfachen Alarmsynchronisationseinrichtung (ASE), die Bestandteil der Prozeßanschlußeinheit der PTB ist, realisiert werden.

Diese in der Abbildung dargestellte Einrichtung liefert

4 "Basisalarmeingänge", die gleichermaßen für Organisations-Daten- oder Kettungsalarme verwendet werden können. Unter Kettungsalarmen seien dabei die Alarme von nachgeordneten geketteten Anschlußeinheiten verstanden, die dort als Organisations- oder Datenalarme angemeldet wurden. Ein Basisalarmeingang ist durch die 4 Signalanschlüsse A_i, F_i, Q_i und N_i gekennzeichnet.

Durch Setzen des Signaleinganges A_i wird ein Alarm beim Basisalarmeingang i gestellt. Dieser wird, wenn das Freigabesignal F_i und das Abfragesignal des i. Einganges gesetzt sind und keine zentrale Initiative bearbeitet wird (Z $\hat{=}$ logisch 1), als Alarm A in Richtung Rechner durchgeschaltet.

Die Basisalarmeingänge werden mit der Taktgenerator-Zähler-Kombination ständig abgefragt, bis ein zu bearbeitender Alarm gefunden wird. Der Zähler besitzt 4 Zählzustände, die durch das Einzählen der Taktimpulse zyklisch durchlaufen werden. Mit jedem Zählzustand werden ein bestimmtes Abfragesignal gesetzt und der damit verbundene Basisalarmeingang abgefragt. Bei Erfüllung der dreifach-UND-Bedingung (s. Abb.) werden die Taktimpulse gesperrt, der Zählzustand festgehalten und der anstehende Alarm bearbeitet.

Mit dem Durchschalten eines Alarmes in Richtung Rechner wird das Aufschaltsignal N_i des betreffenden Basisalarmeinganges gesetzt.

Mit diesem wird die Alarmbegleitinformation auf den Bus zum Rechner geschaltet, also bei einem Organisations-

alarm die Alarmnummer, bei einem Dateneingabealarm das Datenwort und bei einem Kettungsalarm die Information vom Kettungsbus zu den geketteten Prozeßanschlußeinheiten.

Über Q_i wird die vom Rechner kommende Quittung Q zum alarmstellenden Gerät weitergereicht, das daraufhin den Alarm zurücknehmen muß. Dadurch wird die Schaltung wieder in den Abfragezustand versetzt.

Wird die Anforderung im Fehlerfall nicht zurückgenommen, verschwindet sie durch die R-C-Kombination nach einer angemessenen Zeit von selbst.

Mit Hilfe des Freigabesignals F_i kann der Basisalarmeingang zum Zwecke der Programmkoordinierung gesperrt oder geöffnet werden. Bei Datenalarmen wird z.B. gesperrt, wenn der Datenbereich für den Blockverkehr nicht freigegeben ist.

Die Alarme werden verzögert zum Rechner weitergeleitet. Diese Verzögerung (s. Abb.) hat zwei Gründe:

1. wird dadurch sichergestellt, daß immer die Alarmbegleitinformation an der EA-Schnittstelle anliegt, wenn ein Alarm erscheint; das Aufschaltsignal N_i wird vor dem Alarm A gesetzt,

2. werden dadurch Alarme aufgehalten, für die eine Sperrbedingung z.B. durch Signallaufzeiten verspätet eintrifft, so daß auch diese nicht zur EA-Schnittstelle gelangen können.

Nach dieser allgemeinen Beschreibung der ASE soll nun ihre

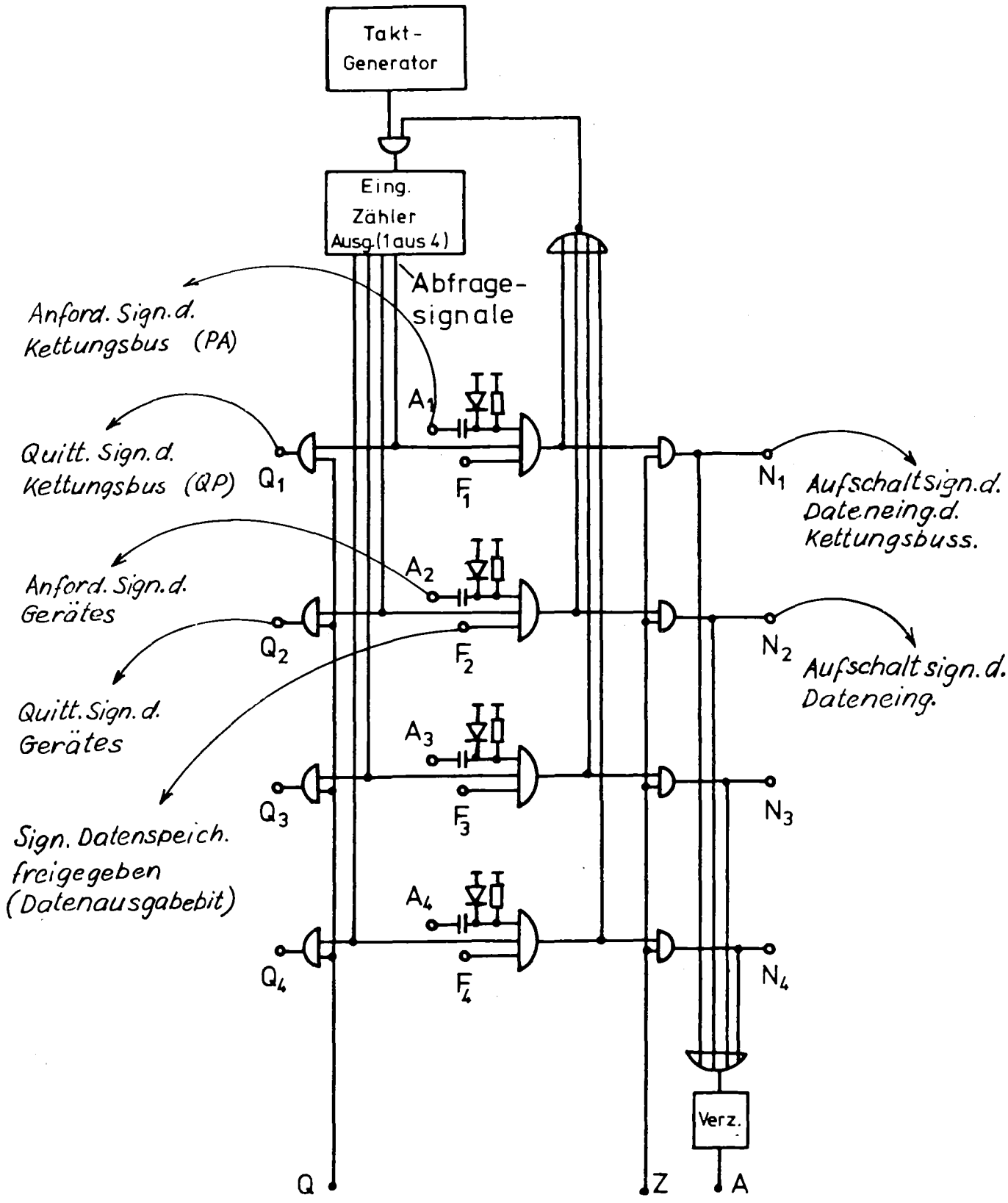
Anwendung an einem praktischen Beispiel verdeutlicht werden. Und zwar soll über den Basisalarmeingang 2 ein Eingabe-Blockverkehr stattfinden. Dazu sind auf dem internen Programmierfeld der EA-Steuerung über Steckleitungen die folgenden in der Abbildung verzeichneten Signalverbindungen herzustellen:

A_2 ist mit dem Anforderungssignal des peripheren Gerätes, das den Blockverkehr wünscht, zu verbinden und Q_2 mit dem Quittungssignal des Gerätes. Der Freigabesignaleingang F_2 wird mit einem Signal beschaltet, das immer dann gesetzt ist, wenn der Datenpuffer im Arbeitsspeicher freigegeben und noch nicht vollgeschrieben wurde. Dieses Signal kann z.B. durch ein normales Ausgabebit realisiert werden. N_2 wird mit dem Aufschaltsignal des Dateneinganges verbunden, an dem die zu übertragende Information ansteht. Dieser Dateneingang ist einer von jenen, die auch für zentrale Initiativen benutzt werden und daher in der Prozeßanschlußeinheit sowieso vorhanden sind. Er wird lediglich durch das Aufschaltsignal N_2 statt durch ein zentral gesteuertes Adressierungssignal aufgeschaltet. Im Falle, daß an eine EA-Steuerung noch weitere EA-Steuerungen gekettet sind, muß auch ein Basisalarmeingang als Kettalarmeingang programmiert werden. Über diesen Alarmeingang werden alle Alarme der nachgeordneten geketteten EA-Steuerungen synchronisiert. Diese Alarme können Organisations- oder Datenalarme sein.

Auf der Abbildung wurde der Basisalarmeingang 1 als Kettalarmeingang auf dem internen Programmierfeld beschaltet. Dazu sind lediglich A_1 und Q_1 mit dem Alarm- bzw. Quittungssignal des Kettungsbusses zu verbinden, der zu den nachgeordneten geketteten Anschlußeinheiten führt. N_1 ist mit

dem Aufschaltsignal der Dateneingänge dieses Kettungsbusses zu beschalten. Das Freigabesignal F_1 wird nicht benötigt. Die verbleibenden 2 Basisalarmeingänge sind noch frei für Organisations- oder weitere Datenalarmeingänge. Bei Bedarf kann die ASE durch zusätzliche Basisalarmeingänge ergänzt werden.

Abschließend sei das Ergebnis noch einmal zusammengefaßt. Durch die auf der Abbildung dargestellte einfache Schaltung und die eingezeichneten Signalverbindungen mit Funktionsbausteinen, die für zentrale Initiativen in der EA-Seuerung sowieso vorhanden sein müssen, kann schon ein relativ komplizierter Fall gelöst werden, nämlich der gleichzeitige Blockverkehr über mehrere EA-Steuerungen. Auf die gleiche Weise lassen sich auch beliebige andere und komplexe Alarmkonfigurationen einfach realisieren. Das wird erreicht durch eine systematische Gliederung der gesamten EA-Steuerung in Funktionsbausteine, die auf dem internen Programmierfeld untereinander beliebig verbunden werden können.



Alarmsynchronisationseinrichtung für 4 Basisalarmeingänge

G. Grüßinger
Universität Karlsruhe

T. Wenzel
SIEMENS AG Karlsruhe

BASIC-Interpreter für S310 und R-Maschinen

April 1978

1. BASIC-Interpreter für die SIEMENS 310

1.1 Allgemeines

Gegenstand dieses Berichts ist der BASIC 310-Interpreter, welcher im Auftrag der Fa. SIEMENS am Lehrstuhl für Prozeßinformatik der Universität Karlsruhe entwickelt wurde. Im folgenden sollen eine kurze Übersicht über den Sprachumfang sowie einige Anmerkungen zur Implementierung gegeben werden.

BASIC ist eine problemorientierte Sprache, die auch von Benutzern mit geringen Programmierkenntnissen rasch erlernbar ist. BASIC 310 ermöglicht den schrittweisen Aufbau eines Programms im Dialog und den Programmtest einschließlich Korrektur. Das Programm selbst wird interpretativ abgearbeitet.

1.2 Sprachumfang

Die in Anhang 1 angegebene Liste gibt einen Überblick über die Sprachelemente.

1.2.1 Datentypen und Standardfunktionen

Es existieren nur die Datentypen Real und String, insbesondere also kein Typ Integer. Der Anwender kann ein Datum vom Typ Real (rechnerinterne Darstellung: zwei 16-Bit-Wörter) auch als Binärgröße interpretieren (Bitmuster 32 Bit).

Für Binäroperationen besteht die Möglichkeit, Konstanten in hexadezimaler oder binärer Schreibweise zu notieren.

Bei den Standardfunktionen erlaubt die große Zahl von Stringfunktionen eine leistungsfähige Manipulation von Zeichenketten. Im einzelnen sind möglich:

SUBSTR - Teilstringbearbeitung
VAL - Konvertiert einen String in eine Realzahl
RSTR - die zu VAL inverse Funktion
POS - Position eines Teilstrings in einem String
LENGTH - Länge eines Strings

Die Zeitfunktionen DATE, TIME und CLK stellen eine wichtige Ergänzung für die Anweisungen zur Prozeßbearbeitung dar.

1.2.2 Anweisungen

Für alle Anweisungen gilt: jede Anweisung beginnt mit einer Zeilennummer; pro Zeile ist nur eine Anweisung zulässig. Ein BASIC-Programm besteht dann aus Anweisungen, die nach ihrer Zeilennummer aufsteigend sortiert sind.

Die Wertzuweisung an eine Variable kann auf verschiedene Arten erfolgen: bei LET geschieht dies durch Auswerten eines Ausdrucks, INPUT fordert eine Dateneingabe durch den Anwender an, READ liest den Wert aus einem Datenbereich, welcher mit DATA definiert wurde. Die Manipulation des Datenzeigers mit RESTORE erlaubt mehrmaliges Lesen der Daten.

PRINT ermöglicht die Druckausgabe entweder in einem freien Format oder in einem durch die FORMAT-Anweisung näher spezifizierten Format.

GOTO ... OF erlaubt es, aus mehreren Sprungzielen (Zeilennummern) eines auszuwählen - in Abhängigkeit vom Wert eines Ausdrucks. Dies gilt sinngemäß auch für GOSUB ... OF.

Die Befehlsfolge eines Unterprogramms wird mit'SUB eingeleitet, bei RETURN erfolgt ein Rücksprung auf die auf die GOSUB-Anweisung folgende Zeile.

DEF definiert eine ein- oder mehrzeilige Funktion.

IF ERR ... THEN erlaubt die Programmverzweigung in Abhängigkeit von aufgetretenen Laufzeitfehlern.

Der Anwender hat die Möglichkeit, in Assembler geschriebene Unterprogramme in das BASIC-System mit einzubinden und über CALL aufzurufen (Parameterübergabe möglich).

Ein- und zweidimensionale Felder können mit Hilfe der DIM-Anweisung reserviert werden.

Für den Ablauf segmentierter Programme stehen die Anweisungen COM, TRANS und LINK zur Verfügung. Jedes BASIC-Programm kann als Segment betrachtet werden, LINK und TRANS bewirken einen Segmentwechsel. Ein Datenaustausch über Common-Variable ist möglich (COM).

Die Anweisungen für das File-Handling erlauben die Datenein- oder ausgabe von bzw. auf einen zuvor deklarierten File (d.i. ein Gerät bzw. eine Datei auf Floppy-Disk oder Plattenspeicher), in beschränktem Maße wahlfreien Zugriff auf eine Datei (RESET), Maßnahmen beim Lesen oder Schreiben über das Dateiende (IF EOF ...), Dateilängen zu ändern(ALTER) und Dateien zu löschen (KILL).

Für die Prozeßverarbeitung stehen Anweisungen zum An- und Abmelden von Alarmen, sowie zur Definition von Alarmroutinen zur Verfügung. Parametrierbare Anweisungen zur Digitalein-/ausgabe und Analogein-/ausgabe sind ebenfalls vorhanden.

Ein hoher Testkomfort wird insbesondere dadurch erreicht, daß ein Teil der eben genannten Anweisungen als sogenannte Tischrechneranweisungen Verwendung finden können. Solch eine Anweisung wird ohne Zeilennummer notiert und sofort inter-

pretiert. So können z.B. mit der PRINT-Anweisung nach einem Programmfehler Variablenwerte ausgegeben werden, um so Rückschlüsse auf das Fehlverhalten des Programms ziehen zu können.

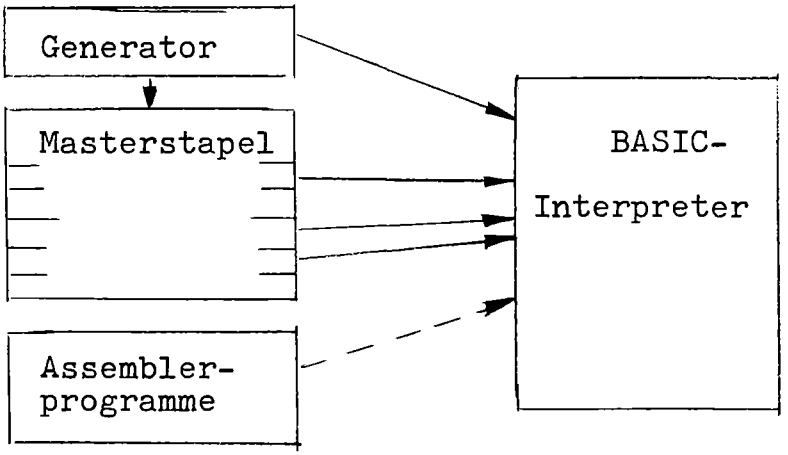
1.2.3 Kommando

Für den Programmaufbau, Programmlauf, das Testen und Korrigieren stehen eine Reihe von Kommandos zur Verfügung, sie sind im Anhang 2 zusammengefaßt.

1.3 Handhabung des BASIC 310

Zur Handhabung des BASIC-Systems ist zu sagen, daß der Anwender nicht etwa ein starres System in die Hand bekommt, sondern vielmehr den Funktions- und Sprachumfang durch eine "Generierung" seinen Anforderungen anpassen kann. Hierbei gibt der Anwender die gewünschte Gerätekonfiguration (Standard- und Prozeßperipherie), den benötigten Hauptspeicher-Ausbau und den Sprachumfang an. Die erforderlichen Systemkomponenten sind:

- Generator urladefähig oder
  Generator unter ORG ladefähig
- BASIC 310 - Masterstapel, bestehend aus Grundsprache - Modulen (siehe Bild).



Der Generator wählt die vom Anwender gewünschten Module aus dem Masterstapel und bindet sie - gegebenenfalls mit zusätzlichen Assemblerprogrammen - zum fertigen BASIC-Interpreter.

Der modulare Aufbau des Gesamtsystems war in der Erstellungsphase von Vorteil, da die einzelnen Komponenten unabhängig voneinander realisiert und getestet werden konnten. Die Implementierung erfolgte auf einer SIEMENS 330.

1.4 Einsatzgebiete des BASIC 310

Für das BASIC 310 haben sich 3 Haupteinsatzgebiete herausgestellt, und zwar:

- Laborautomatisierung
- Arbeitsvorbereitung/Arbeitsüberwachung
- Wissenschaftlich-technische Aufgaben

Bei der Laborautomatisierung werden folgende Problemkreise bearbeitet:

- Labordatenerfassung
- Prozeßdatenerfassung und Verarbeitung
- Erstellung von LS zur Werkzeugmaschinensteuerung
- Röntgen-Fluoreszensanalyse

In der Arbeitsvorbereitung und Arbeitsüberwachung wird in den Werkstätten BASIC 310 benutzt. Dabei handelt es sich um die Anwendungen

- Regressionsrechnung
- Bauplanerstellung
- Arbeitsplanung
- Terminverfolgung, Buchung, Be- und Entlastung

Bei den wissenschaftlich-technischen Aufgaben stehen Lösungen allgemeiner problembezogener Aufgaben im Vordergrund. Hier wird BASIC deshalb eingesetzt, weil diese höhere Programmiersprache folgende Vorteile besitzt:

- dialogorientiert
- leicht erlernbar
- Integrierter Editor, daher leicht änderbar
- Integrierte Testhilfe und
- komfortables Stringhandling

2. BASIC 300, ein neues Konzept

Für die Systeme R10 bis R40 wird ein neuer BASIC-Interpreter und Compiler B300 zur Auslieferung kommen. Dieser hat einige wesentliche Änderungen gegenüber BASIC 310, betreffend:

- Programmstruktur
- Sprachumfang

2.1 Programmstruktur B300 (Bild 1)

B300 besteht aus zwei Komponenten:

- Komponente 1 = Interpreter
- Komponente 2 = Compiler

2.1.1 Interpreter

Er gewährleistet, wie schon bei BASIC 310, volle Interaktivität und bietet somit dem Anwender einen hohen Komfort zur Erstellung und Test seiner Programme.
Als neu ist die Programmstruktur des Interpreters zu sehen. Er ist als Multi-User-Interpreter ausgelegt, d.h. mehrere Benutzer können gleichzeitig arbeiten. Dies wird durch die Teilung des Interpreters in ein Funktionsprogramm und ein Rahmenprogramm erreicht (Bild 2).

Funktionsprogramm

Der funktionelle Teil des Interpreters besteht aus reentrant geschriebenen Bausteinen, die die einzelnen BASIC-Anweisungen bearbeiten. Laufzeitkritische Bausteine befinden sich als CD im Hauptspeicher, laufzeitunkritische Bausteine werden als Segmente in einer Arbeitsdatei auf einen Peripherspeicher abgelegt.

Rahmenprogramm

Das Rahmenprogramm enthält einen Codebereich, Pufferbereich für Daten und Anweisungen, einen Segmentbereich für die Arbeitsdatei und den Anwenderbereich. Es bearbeitet alle Bedienungen und koordiniert den Zugriff zum Funktionsprogramm.

Ablauf des Interpreters

Pro Benutzer ist ein Rahmenprogramm zu laden. Der Ladebinder stellt die Kopplung zwischen Funktionsprogramm und Rahmenprogramm her. Alle Rahmenprogramme müssen mit gleicher Priorität geladen werden. Ein Programmwechselaufruf, der über alle Laufbereiche wirkt, ermöglicht die gleichmäßige Verteilung der CPU-Zeit des Rechners auf alle Benutzer.

Platzbedarf des Interpreters

Der Hauptspeicherplatzbedarf beträgt für das Funktionsprogramm als CD 16K - Worte. Für das Rahmenprogramm sollten vom Benutzer, um ein sinnvolles Arbeiten zu ermöglichen, mindestens 8K Worte zur Verfügung gestellt werden.
Unter diesen Voraussetzungen können bei Anlagen mit einem Arbeitsspeicherausbau von 64K Worten und einem Hauptspeicherbetriebssystem (ORG 300 H, HV) bis zu vier HRP's (= Rahmenprogramme) geladen werden. Bei einem Peripherspeicherbetriebssystem (ORG 300 P, PV) werden die Rahmenprogramme als PRP geladen. Damit sind theoretisch beliebig viele Benutzer erlaubt.

2.1.2 Compiler

Der Compiler ist nicht eigenständig. Nach der Programmerstellung und dem Test eines BASIC-Programms unter dem Interpreter, kann das Programm mit dem Compiler in Grundsprache

überführt werden. Das so compilierte Programm ist sodann unter der Kontrolle eines ORG's direkt abläufig. Dadurch werden die Programmlaufzeiten verkürzt, und es ist ein Multiprogrammig-Betrieb von BASIC-Programmen möglich.
Die Library des Compilers enthält hierzu über CALL aufrufbare Koordinierungsfunktionen des ORG's (z.B. Warten, Fortsetzen usw.).
Der Compiler ist ein segmentiertes Programm und benötigt einen HSP-Bereich von ≥8K Worten.

2.2 Sprachumfang des BASIC B300

Gegenüber dem Sprachumfang des BASIC 310, wurden wesentliche Erweiterungen aufgenommen.

- Datentypen
  Neu sind die Datentypen Integer und Real double, sodaß jetzt folgende vier Datentypen zur Verfügung stehen.

Integer	16 Bit	
Real (single)	32 Bit	6-stellige Genauigkeit
Real (double)	64 Bit	16-stellige Genauigkeit
String	n*8 Bit	

- Funktionen
  Neu bei den Funktionen sind

Bitfunktionen	Bit testen
	Bit setzen
	Bit löschen
Stringfunktionen	Umwandlung von ASCII-Zeichen in ihr Dezimaläquivalent und umgekehrt

- Matrizenanweisungen
  Diese Anweisungen wurden neu aufgenommen und zwar handelt es sich um:

  Ein- und Ausgabe von Matrizen

Addition, Subtraktion, Multiplikation von Matrizen
Skalare Multiplikation
Transponieren, Invertieren
Einheits-, Eins- und Nullmatrix bilden
Determinate bestimmen, Redimensionieren

- Ein-/Ausgabe

  Sie wurde um eine vom Anwender steuerbare formatierte Ein-/Ausgabe erweitert. Dadurch kann sich der Anwender an beliebige Datenstrukturen anpassen.

- Unterprogrammnahtstelle

  Mit der neuen Unterprogrammnahtstelle, die gleich wie in Fortran und Cobol sein wird, ist es möglich, Unterprogramme, die in diese Sprache oder im Assembler ASS300 erstellt wurden, in BASIC-Programme einzubinden. Genauso ist es dann möglich, BASIC-Unterprogramme in eines der mit der oben genannten Sprache erstellten Programme, einzubinden.

Neben diesen hier aufgeführten Spracherweiterungen wurden auch bei anderen Sprachstatements Änderungen durchgeführt, die ein besseres Arbeiten mit BASIC ermöglichen. Dies gilt auch für einige Kommandos.

Trotz dieser Erweiterungen und Änderungen wurde erreicht, daß mit BASIC 310 erstellte Programme auch unter BASIC B300 ablauffähig sind. Diese Kompatibilität ist nur in wenigen Punkten nicht möglich, so z.B. bei Programmen mit Prozeß-Ein-/Ausgabenanweisungen.

SPRACHELEMENTE

Datentypen

- Real/Binär (Bereich $\pm 10^{-38} \ldots \pm 10^{38}$)
- String (max. 251 Zeichen)

Variablen

- Real/Binär
- String

Konstanten

- Zahlenkonstanten
- Stringkonstanten
- Hexadezimalkonstanten
- Binärkonstanten

Operatoren

- arithmetische (+, -, *, / , **)
- Vergleichsop. (<, <=, =, >,>=,#)
- logische (NOT, AND, OR, XOR)
- String op. (&)

Standardfunktionen

- mathematische (SIN, COS, ..., LOG, SQR, ...)
- Stringfktn. (SUBSTR, VAL, RSTR, POS, LENGTH)
- Zeitfktn. (DATE, TIME, CLK)

Ausdrücke

- arithmetische
- Stringausdrücke

Anweisungen

allgemein

LET	Wertzuweisung
INPUT	Wertzuweisung
READ	Wertzuweisung
DATA	Datenbereich
RESTORE	Datenzeiger rücksetzen
PRINT⊄FORMAT⊅	(formatierte) Druckausgabe
FORMAT	Ausgabeformat
GOTO⊄ ... OF⊅	Sprunganweisung
GOSUB⊄ ... OF⊅	Unterprogrammsprung
SUB	Unterprogramm
RETURN	UP-Rücksprung
DEF	Funktionsdefinition
FOR ... TO ⊄... STEP⊅	Programmschleife
NEXT	Ende einer Programmschleife
IF ... THEN	bedingte Programmverzweigung
IF ERR ... THEN	Progr.-verzweigung - Laufzeitfehler
END	Programmende
STOP	Programmausführung stoppen
REM	Kommentar
CALL	Aufruf Assembler - UP
RAD	Umschaltung Bogenmaß
DEG	Umschaltung Altgrad
GRAD	Umschaltung Neugrad
SHIFT	zyklisches Schieben
DIM	Feldreservierung
COM	Commonliste
TRANS	Segmentwechsel
LINK	Segmentwechsel mit RSP

⊄...⊅ wahlweise

File-Handling

FILE	File-Deklaration
WRITE	Datenausgabe
READ	Dateneingabe
RESET	Setzen des Dateizeigers
IF EOF ... THEN	Progr.-verzweigung bei END-OF-FILE
ALTER	Dateilänge ändern
KILL	Dateien löschen

Prozeßbearbeitung

ON INT ... THEN	Alarmanmeldung
INT	Alarmroutine
INTEND	Ende einer Alarmroutine
DISABLE	Alarme abmelden
ENABLE	Alarme wiederanmelden
WAIT	Programm anhalten
DIN	Digitaleingabe (dyn. + stat.)
DOUT	Digitalausgabe
AIN	Analogeingabe (integr. + mom.)
AOUT	Analogausgabe

INTERPRETER - KOMMANDOS

NEW	Anfangszustand
EDIT	Edit - Modus
LIST	Programmzeilen ablisten
DELETE	Programmzeilen löschen
TRACE	Programmablaufüberwachung
RENUM	Neunumerierung
AUTO	Zeilennummernvorgabe
RUN	Programm starten
CONT	Programm forsetzen
BYE	Interpreter beenden
BDG/PRG	Wahl des Bedien-/Ausgabegerätes
SIG	Zeilensteuerung
SAVE	Programm archivieren
OLD	Archiviertes Programm lesen
MERGE	Programm(stück) einfügen
COPY	Kopieren
CAT	Dateienverzeichnis ausgeben

BASIC B300 für die Systeme R10-R40

Bild 1: Programmstruktur-Übersicht

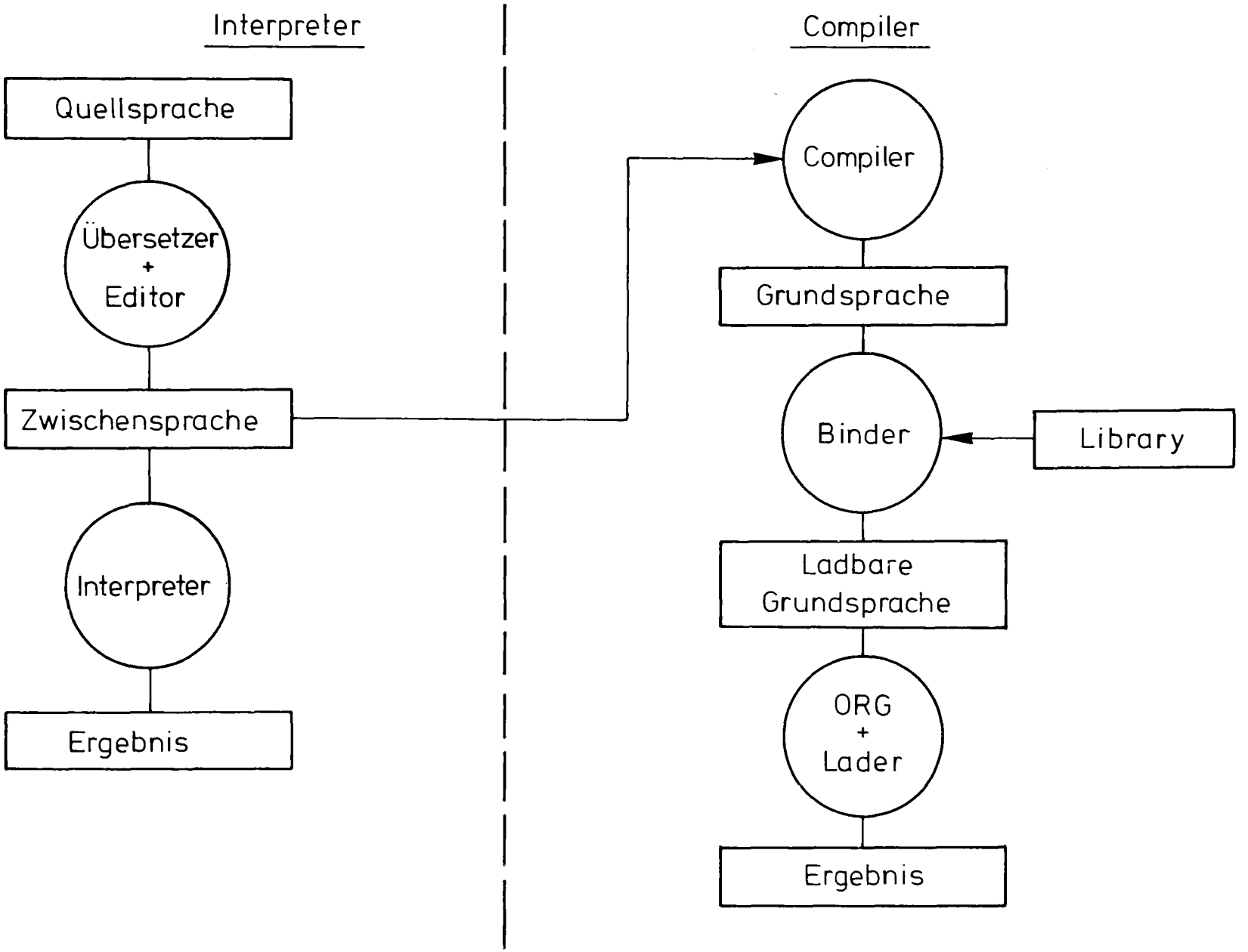
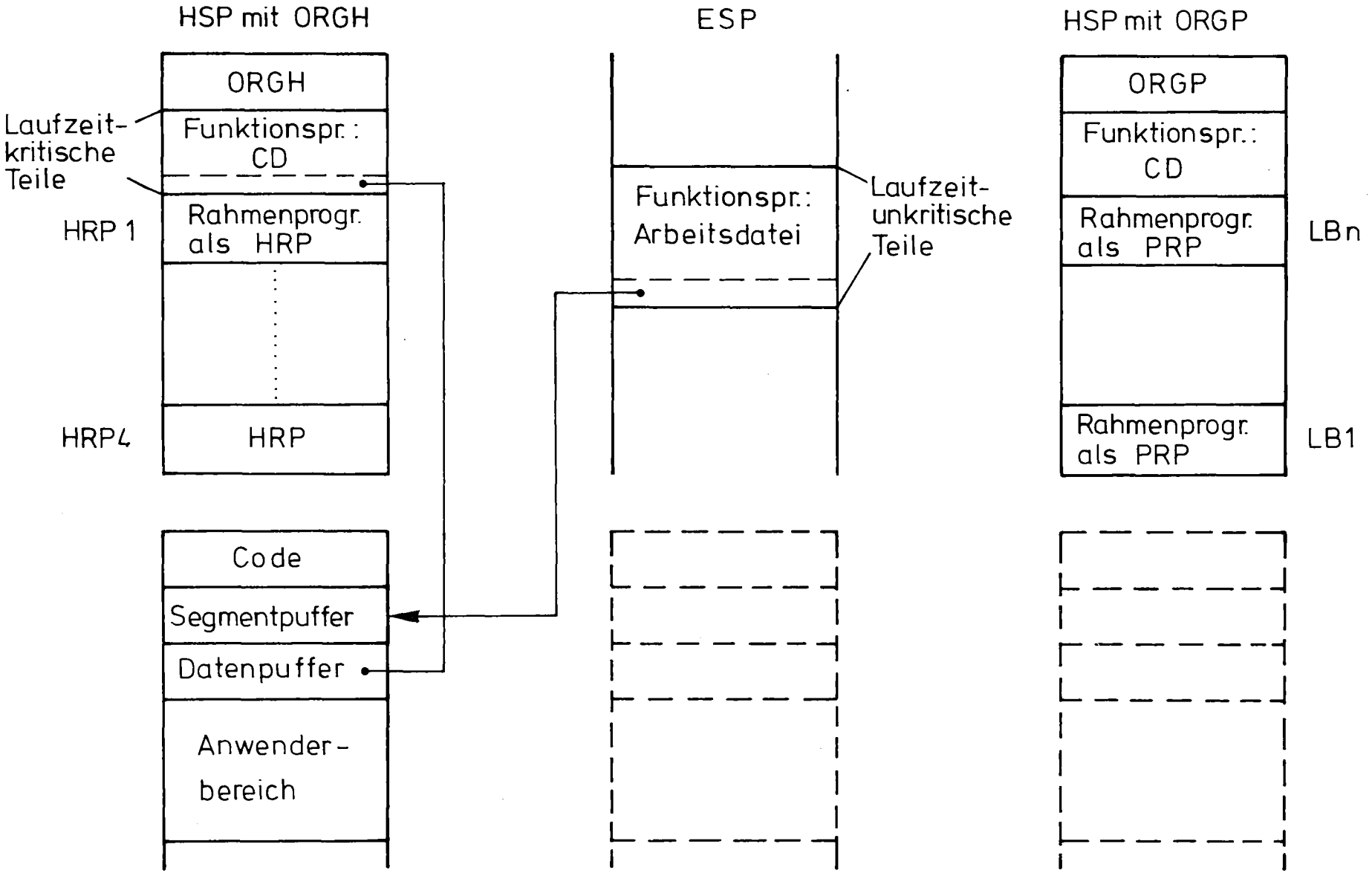


Bild 2: Struktur des Multi-User-Interpreters



Rahmenprogramm = Benutzer = HRP oder PRP

B. Uhlmann

Erweiterte Datei-Organisation für den S310-BASIC-Interpreter

Die Arbeiten hatten zum Ziel, die vorhandene Dateistruktur auf Floppy-Disk der S310, wie sie der BASIC-Interpreter benutzt, zu erweitern. Diese Erweiterungen sind zunächst speziell für den Interpreterbetrieb durchgeführt. Zwar sind die Standard-Dateifunktionen Ergebnis von ORG310-Komponenten im Interpreter, und Erweiterungen wären prinzipiell auf dieser ORG-Ebene wünschenswert gewesen, aber mangelnde Arbeitsunterlagen haben bisher nur zu sog. Nutzerprogrammen für den BASIC-Interpreter geführt.

Die Standard-Dateifunktionen umfassen das Einrichten, Löschen, Umbenennen, Verkürzen und Verlängern sequentiell schreib- und lesbarer Dateien. Die Dateiattribute (Name, Länge, Adresse) werden dabei in einem Buchhalter geführt.
Die im folgenden beschriebenen Erweiterungen betreffen alle nur die interne Organisation und den Zugriff zu bestehenden Dateien im Sinne der Standardfunktionen. Damit üben die Erweiterungen keinen Einfluß auf den Buchhalterinhalt aus. Die Dateien werden mit herkömmlichen Aufrufen eingerichtet, ggf. verändert oder gelöscht. Nur der Datentransfer und der interne Aufbau der Dateien erhält zusätzliche Möglichkeiten :

1. Wahlfreier Zugriff, direkt und ungepuffert.
   Lesen/Schreiben mit Disk-Adreß-Angabe.

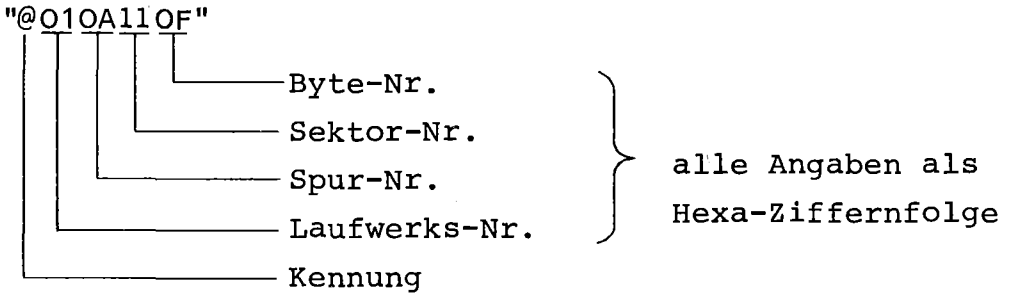
2. Indexsequentielle Organisation
   über Indextabelle mit Schlüssel und Adresse.

3. Zyklische Dateien
   als Umlaufspeicher (First In - First Out)

Die Darstellung der Daten ist kompatibel zur Standard-Datei (eine indexsequentielle Datei kann beispielsweise rein sequentiell mit der Standardfunktion READ gelesen werden).

Wahlfreier Zugriff

Bei diesem Zugriff zur Floppy-Disk wird ohne Zwischenpufferung direkt zwischen der Floppy-Disk-Anschaltung und dem Variablenbereich des Interpreters transferiert. Die Adresse auf der Diskette (Spur-Nr., Sektor-Nr., Byte-Nr.) ist vom Anwender-Programmierer selbst anzugeben! Dies muß eine BASIC - Zeichenkettenvariable mit folgendem Aufbau sein:



Damit erhält der Anwender-Programmierer leider (zur Zeit noch) die Möglichkeit seine Datenbestände auf der Diskette unabhängig von Dateigrenzen zu beeinflussen. Hier ist also größte Sorgfalt und Disziplin zu wahren.
Bei einem wahlfreien Zugriff wird die Adressvariable immer mitgeführt, d.h. nach dem Zugriff beinhaltet sie die Adresse desjenigen Bytes, das auf das zuletzt zugegriffene Datenelement folgt.
Für den erstmaligen Zugriff zu einer Datei steht ein spezieller Aufruf (POSE) zur Verfügung, der die notwendige Adresse liefert. Der Zugriff selbst erfolgt durch Aufruf der beim Generieren des Interpreters mit einzubindenden sog. Nutzerprogramme mit den Namen GET (Leseoperation) und PUT (Schreiboperation).

Beispiel

```
CALL PUT (A$,B(*,*),C$(*))
          |     |     |
          |     |     +---- Zeichenketten-Feld
          |     +---------- Zahlenfeld (2-Dim)
          +---------------- Adressvariable
```

Die Gasamtzahl der Parameter darf 8 nicht überschreiten.
Außer der Adressvariablen sind nur Felder als Parameter zugelassen.
Bei Zeichenkettenfeldern wurde für die Aufgaben in Datenhaltungssystemen folgende zusätzliche Eigenschaft eingebaut:
Wenn ein zu übertragendes Feldelement den formalen Aufbau einer Adressvariablen hat (@...), wird sie auch als solche behandelt. Zunächst wird sie aber normal mit übertragen, dann erst wird sie zur sofortigen Neupositionierung benutzt. Danach wird an der neuen Position mit dem Datentransfer fortgefahren. Damit wird es möglich, vorhandene Datenbestände zu einem späteren Zeitpunkt fortzusetzen.

T$(O) bis T$(N-1) enthalten alte Daten
T$(N) = " @... "
T$(N+1) bis T$(K) erhalten die Fortsetzung.

Indexsequentielle Dateien

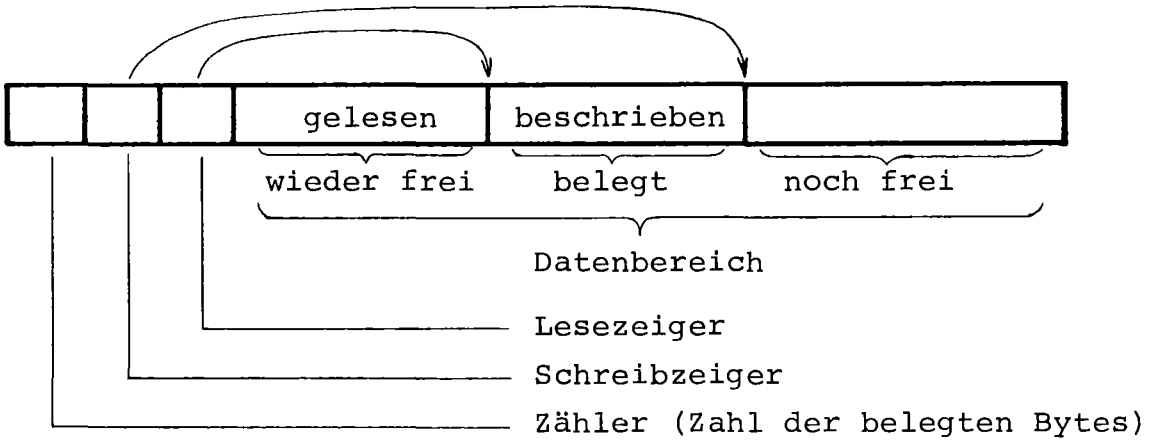
Während beim wahlfreien Zugriff die Adresse auf der Diskette bei jedem einzelnen Zugriff beliebig angegeben werden konnte, aber immer angegeben werden mußte, ist jetzt die Frage der Adress-Herkunft anders geklärt: Die sogenannte Indexliste enthält jeweils paarweise einen Ordnungsbegriff (Schlüssel) und eine Disk-Adresse. Für einen Zugriff wird aus der Indexliste die zum Schlüssel gehörende Startadresse benutzt, um dort beginnend sequentiell auf die Diskette zuzugreifen.

Die bereits erwähnten Aufrufe GET und PUT leisten dies dadurch, daß sie zunächst den Zugriff zur Startadresse wahlfrei ermöglichen und nach dem Zugriff eine weitergestellte Adresse liefern. Ein folgender Zugriff trifft also auch das folgende Datenelement.
Damit ist sowohl wahlfreier als auch indexsequentieller Betrieb mit den gleichen Programmen möglich. Dies gilt in gleicher Weise für den nächsten Dateityp.

Zyklische Dateien

Diese Dateien werden für alle Formen der peripheren Warte-bereiche benötigt. Sie kann man auch als FIFO - Dateien bezeichnen.
Neben dem Dateiinhalt wird diese Datei durch zwei Adressvariable gekennzeichnet. Diese geben an, ab wo die Datei frei ist, also beschrieben werden kann und zum anderen, ab wo sie beschrieben worden ist, also gelesen werden kann.

FIFO-Datei :



Wenn durch Schreiben oder Lesen das Ende der Datei erreicht ist, wird wieder von vorn begonnen. Die Datei ist voll, wenn der Schreibzeiger den Lesezeiger einholt , und sie ist leer, wenn der Lesezeiger den Schreibzeiger einholt.

Die zusätzliche Zählervariable dient der Überprüfung vor dem Schreiben, ob noch genügend Platz ist, bzw vor dem Lesen, ob überhaupt Inhalt vorhanden ist. Sie ist auch notwendiges Kriterium, um volle von leerer Datei zu unterscheiden, wenn beide Zeiger gleich sind.

Auch diese Dateiform kann mit den Zugriffs-Aufrufen GET und PUT bearbeitet werden, da sowohl Lesezeiger als auch Schreibzeiger durch die betreffende Operation weitergestellt werden. Bei diesen Ausführungen mag vielleicht deutlich werden, daß die vermeintliche Gefahr der Zerstörung von Datenbeständen durch fehlerhafte Anwenderprogramme nicht so groß ist. Der wirklich wahlfreie Zugriff wird eher die Seltenheit sein. Die Disk-Adressen werden also häufig vom Anwender gar nicht manipuliert. Beim Organisieren einer zyklischen Datei muß der Anwender allerdings die Datei-Endeadresse nach jedem Zugriff mit dem jeweiligen Zeiger vergleichen und diesen ggf. zurück auf den Dateianfang setzen.

Realisierung der Aufrufe

Die drei notwendigen Programme POSE, GET und PUT sind in 310-Assembler-Sprache geschrieben und werden beim Generieren des Interpreters als Nutzer-Unterprogramme eingebunden. Sie beanspruchen ca. 820 Worte statisch. Zur Verwendung ist natürlich notwendig, daß darüberhinaus beim Generieren die BIBUS (Bibliotheks-Unterprogramme) für Zeichenkettenverarbeitung, CALL-Bearbeitung und Dateiverwaltung mit eingebunden werden.

Unverträglichkeiten ?

Da die Aufrufroutinen ohne ORG-Baustein-Benutzung geschrieben wurden, also der gesamte Peripherieverkehr "am ORG vorbei" geht, aber doch eine Anschlußstelle benutzt wird, die auch

vom ORG benutzt wird, wurde konsequent darauf geachtet, daß nach dem Zugriff mit GET oder PUT alle Zellen der Registertafel der Anschlußstelle und alle Register in der Anschaltung selbst so restauriert werden, wie sie vor dem Zugriff aussahen. Da genaue ORG-Kenntnisse nicht verfügbar waren und Arbeitsunterlagen über die Floppydisk-Anschaltung nicht alle Fragen klären konnten , war nur vom Betrieb Auskunft zu erlangen, ob Konflikte auftreten. Dies war nach einigen Korrekturen nicht mehr der Fall.

Das System arbeitet seit ca.1 Jahr für ein Datenhaltungssystem in einem klinisch-chemischen Labor.

FLOARP

EIN FLOPPY-ARBEITSPROGRAMM

FÜR DIE FLOPPY-DISK AN DER SIEMENS 330

Kernforschungsanlage Jülich GmbH
Zentralinstitut für Angewandte Mathematik

L. Loup

VORWORT

Das FLOARP ist ein Arbeitsprogramm, das dem Anwender die Möglichkeit gibt, auch an der Siemens 330 die Floppy-Disk nicht nur als rein seriell arbeitendes Lochstreifen-Lese/Stanzgerät, sondern als Kleinplattenspeicher mit Direktzugriff zu benutzen.

Dies wurde realisiert durch das Einführen einer Dateiorganisation, die als Buchführung auf jeder Diskette abgelegt wird. Da diese Dateiorganisation in den wesentlichen Punkten übereinstimmt mit der, die auf DEC Maschinen unter dem System RT-11 vorliegt, ist auch noch eine Kompatibilität zwischen der PDP-11 und der Siemens 330 gegeben.

Technische Daten

Hardware:	ZE 330
	PLSK
	BTAE
	BDRA
	DRUA
	FLOP
Software:	ORG 330 PPII
ORG-Funktionen:	BEDIEN,BEENDE,SIMKON,ZEIT,ZUTEIL
	FLOARP
Sprache:	ASS 300, MAS 300
Speicherbedarf:	HSP 5260 Worte einschließlich aller PT, Puffer und Konstanten

Inhaltsverzeichnis
---

1. Allgemeines

2. Programmbedienung

3. Kommandos

3.1 EINR: Buchführung einrichten
3.2 BUCH: Buchführung protokollieren
3.3 LOE: Elemente löschen
3.4 KOMP: Buchführung komprimieren
3.5 PROT: Elemente protokollieren
3.6 RUN: Elemente umsetzen
3.7 ENDE: Programm beenden

4. Fehlererklärungen

5. RWFLOP Beschreibung des Unterprogramms

6. Beispiel für die Benutzung des RWFLOP

1. ALLGEMEINES

Die Beschreibung soll dem Benutzer von FLOARP die Arbeitsweise des Programms erläutern, die einzelnen Funktionen erklären und die Belegung der Diskette aufzeigen.

Die Floppy-Disk ist auf der Siemens 330, dem System -ORG 330 PPII-, als Lochstreifenleser und -stanzer bekannt. Das heißt: Das Gerät ist beim Generieren als Standardgerät (LSTE,LSTA) eingetragen worden und deshalb auch nur mit Standardaufrufen (Lochstreifenverarbeitung) anzusprechen.
Da es keine Möglichkeit gibt, Standardgeräte zusätzlich selbst programmtechnisch anzusprechen, ist der Anwender gezwungen, die Floppy-Disk dem System als Sonderperipherie bekanntzumachen. Für das FLOARP wird sie als Schellreaktionsperipherie (SNEP) durch den Generierparameter /G:DIRE(anr,0),SNEP(0),,,,,E1;
(siehe ORG-Beschreibung 32.3.3) eingetragen.
Um nicht zwei Versionen generieren zu müssen (LSTE,LSTA und SNEP), kann die Standardverarbeitung als Lochstreifen auch mit FLOARP durchgeführt werden (siehe Kapitel 3.6 Kommando RUN).

Die Anschlußnummer der Floppy-Disk wird bei Programmstart aus dem peripheren Unterbrechungsweichenregister (PUWR) ausgelesen und der Floppy-Anschaltung mitgeteilt. Sind mehrere SNEP-Geräte generiert, so wird die Anschlußnummer über Blattschreiber angefordert.

Das FLOARP ist so aufgebaut, daß man sowohl Daten transferieren, als auch die notwendigen Arbeiten an und mit dem Buchhalter durchführen kann. Im FLOARP selbst sind die Übertragungsmöglichkeiten Floppy-Floppy, Platte-Floppy, Floppy-Platte und Floppy-Drucker gegeben.
Dabei wird auf die Platte im Sinne der Dateiorganisation nach ORG-Normierung zugegriffen. Der Zugriff zur Floppy gleicht in etwa der Bibliotheksorganisation im ORG, ist aber aus Kompatibilitätsgründen mehr an das RT-11 System angelehnt.

Soll von einer Diskette gelesen werden, so wird zuerst die Buchführung nach dem Namen des zu lesenden Elements durchsucht. Ist er eingetragen, wird der Transfer gestartet. Andernfalls wird dem Benutzer mitgeteilt, daß das Element nicht vorhanden ist und auf neue Eingabe gewartet.

Bei dem Beschreiben einer Diskette wird ebenfalls zuerst die Buchführung auf den einzutragenden Namen durchsucht. Ist dieser schon vorhanden, wird eine Fehlernachricht ausgegeben und dem Benutzer freigestellt, den eingetragenen zu löschen oder den neuen umzubenennen. Ist der neue Name in der Buchführung nicht enthalten, dann wird er eingetragen und als vorläufig gekennzeichnet. Erst bei fehlerfreier Beendigung des Transfers wird der Eintrag permanent gesetzt.

Sollen Daten transferiert werden zwischen anderen Geräten und der Floppy, so steht dem Anwender eine als Unterprogramm aufgebaute Routine zur Verfügung, die er mit seinem Programm koppeln und mit der er die Diskette dann nach dem Schema von FLOARP lesen und beschreiben kann (siehe Kapitel 5. RWFLOP).

Um die relativ hohe Speicherkapazität des kleinen Datenträgers (256 K Bytes) im Gegensatz zur Lochstreifenverarbeitung auszunutzen, wurde die Diskette mit einem Buchhalter versehen. Dieser gestattet es, direkt auf die einzelnen Elemente zuzugreifen. Dabei wird nur in ganzen Blockgrößen von 512 Bytes (4 Sektoren) übertragen.

Aus Kompatibilitätsgründen sieht die physikalische Belegung der Diskette (77 Spuren a 26 Sektoren a 128 Bytes) folgendermaßen aus:
Ist innerhalb einer Spur das Ende eines Sektors erreicht, wird beim übernächsten Sektor fortgefahren. Dabei werden zuerst die Sektoren mit der ungeraden Adresse, und dann die mit der geraden Adresse angesprochen. Bei Spurwechsel werden 6 Sektoren übersprungen.

z.B. Spur 1 Sektor 1, 3, 5...21,23,25
2, 4, 6...22,24,26

Spur 2 Sektor 7, 9,11...21,23,25, 1, 3, 5
8,10,12...22,24,26, 2, 4, 6

Spur 3 Sektor 13,15,17...21,23,25, 1, 3, 5, 7, 9,11
14,16,18...22,24,26, 2, 4, 6, 8,10,12

Spur 0 wird nicht benutzt. Ebenso bleiben in der Spur 1 alle Sektoren frei bis auf die beiden (logisch letzten) 24 & 26. Der Buchhalter beginnt bei Spur 1 Sektor 21 (Block 6) und benötigt 32 Sektoren (8 Blöcke). Sofort dahinter (Spur 3 Sektor 21, Block 14) erfolgt in 1920 Sektoren (480 Blöcken mit 245.760 Bytes), die Abspeicherung der einzelnen Elemente.

2. PROGRAMMBEDIENUNG

Die Bedienungsanweisung wird über das Standard-Bedienprogramm abgewickelt und muß deshalb den Konventionen dieser Bedienung entsprechen (:PRGRNR:AUFTRAG:PARAMETER;ETX). Die Kommandosyntax des Auftrages und der Parameter ist angelehnt an die vom System vorgesehene Form. Bei einem Syntaxfehler wird die Kommandozeile bis zur fehlerverursachenden Stelle wiederholt und mit einem Fragezeichen abgeschlossen. Danach wird eine neue Eingabe erwartet.

Für die Geräteparameter ist nach ORG-Normierung die Gerätekennung mit vier Buchstaben vorgesehen (PLSK und FLOP), und für die Gerätenummer nur die logischen Anschlußstellennummern (0) oder (1). Fehlt die Angabe, so wird vom FLOARP 0 gesetzt.

Ein Dateiname (Zugriff auf PLSK) muß der Dateiorganisation des ORG entsprechen und mindestens 1, max. 6 Buchstaben und/oder Ziffern haben, wobei hier das erste Zeichen ein Buchstabe sein muß. Für den Elementnamen (Zugriff auf FLOP) gilt für die ersten sechs Buchstaben das Gleiche wie zuvor, er kann aber aus Kompatibilitätsgründen zur PDP-11 mit einer Erweiterung von drei Buchstaben, die mit einem Punkt eingeleitet wird, versehen werden.

Außer bei den Kommandos PROT: und RUN: brauchen die Kennworte EQ,EB,EL,AQ,AB,AL nicht angegeben zu werden, da ihre Aussagen durch das FLOARP gesetzt sind. Bei den beiden vorgenannten Kommandos haben sie allerdings eine spezielle Bedeutung, die bei den betreffenden Kommandobeschreibungen erläutert wird.
Unterstrichene Parameter können vom Benutzer frei gewählt oder weggelassen werden.

3. KOMMANDOS

3.1 ***** EINR:EB<FLOP(gn)>;etx *****

Mit diesem Kommando wird der Buchhalter auf einer Diskette eingerichtet. Es werden vier Segmente (4096 Bytes) für die Buchführung reserviert und ein Leereintrag mit 480 Blöcken (245.760 Bytes) vorgenommen. Jede Diskette muß, bevor sie zum ersten Mal bearbeitet werden kann, eingerichtet werden und steht danach für alle Arbeiten unter FLOARP zur Verfügung.

3.2 ***** BUCH:EB<FLOP(gn)>ELNAME;etx *****

Es wird das Inhaltsverzeichnis des Buchhalters einer Diskette protokolliert. Mit der Angabe eines Elementnamens wird nur dieses Element, sonst alle eingetragenen Elemente mit der folgenden Information ausgegeben:

Elementname, Anzahl der belegten Blöcke und, wenn vorhanden, das Tagesdatum des Eintrages. Weiter werden die Leereinträge mit Blockanzahl ausgegeben und zum Abschluß die Anzahl der eingetragenen Elemente mit der gesamten Blockzahl, sowie die gesamte noch zur Verfügung stehende Leerblockanzahl.

3.3 ***** LOE:EB<FLOP(gn)>ELNAME;etx *****

Der angegebene Elementname wird aus der Buchführung durch einen Leereintrag gelöscht. Steht unmittelbar davor oder dahinter noch ein Leereintrag, so wird die Anzahl der Leerblöcke zusammengefaßt und dann nur ein Leereintrag vorgenommen. Soll ein ganzer Buchhalter gelöscht werden, so kann dies durch ein erneutes Einrichten der Diskette geschehen.

3.4 ***** KOMP:EB<FLOP(gn)>:AB<FLOP(gn)>;etx *****

Sind im Buchhalter mehrere Leereinträge mit kleiner Blockanzahl vorhanden, dann stehen diese dem Anwender als zusammenhängende Blockzahl für einen Eintrag nicht mehr zur Verfügung. Um dieses zu ändern, kann der Anwender den Buchhalter komprimieren:

Vorhandene Leereinträge werden dann durch Vorschieben der Elemente beseitigt, so daß am Ende ein einziger Leereintrag mit der gesamten Anzahl von Leerblöcken übrigbleibt.

Der Anwender kann sowohl die komprimierte Ausgabe auf die gleiche Diskette, wie auch auf eine andere -wenn ein zweites Laufwerk vorhanden ist- ablegen. Wenn möglich, sollte die letztere Version gewählt werden, denn im Falle eines Übertragungsfehlers bleibt hier die Original-Diskette erhalten, und der Komprimierlauf kann wiederholt werden. Auf diese Weise kann eine Diskette auch dupliziert werden. Beim Komprimieren auf einer Diskette geht im Fehlerfalle das Element, bei welchem der Fehler auftaucht, dem Anwender verloren.

3.5 ***** PROT:EQ<FLOP(qn)>ELNAME;etx *****

Mit diesem Kommando werden alphanumerische Daten des angegebenen Elements auf dem Drucker protokolliert. Das Kennwort EQ hat in diesem Aufruf eine besondere Bedeutung.
Fehlt EQ im Aufruf, so wird der Datensatz als im PDP-11 Format vorliegend interpretiert und als Zeilenende ein LINE-FEED akzeptiert. Das Druckende wird durch ein CTRL/Z eingeleitet.

Das Kennwort EQ gibt an, daß der Datensatz im Siemensformat vorliegt. Als Zeilenende wird nun ein ETX genommen und das Druckende durch /* erkannt.
Ist im Ausgabepuffer kein ETX enthalten, werden die Daten für die Ausgabe auf 80 Zeichen (Kartenformat) pro Zeile geblockt und ausgegeben.

3.6 ***** RUN:EQ<FLOP(qn)>ELNAME:AQ<FLOP(qn)>ELNAME;etx *****

```
    EB                  AB
    EL          XXXXXX  AL          XXXXXX

RUN:EQ<PLSK(qn)>DTNAME:AQ<FLOP(qn)>ELNAME;etx
    EB                  AB
    EQ                  AL          XXXXXX

RUN:EQ<FLOP(qn)>ELNAME:AQ<PLSK(qn)>DTNAME;etx
    EB                  AB
    EL          XXXXXX  AQ
```

Der Datentransfer kann in drei Arten erfolgen. Die Kennbuchstaben Q,B und L geben den vorliegenden Transfertyp an.
Wird beim Kommando die Kennung Q gegeben, so werden die Daten byteweise übertragen. Die Kennung B hat wortweise Übertragung zur Folge, und bei L wird der Transfer im Lochstreifen-Modus durchgeführt.
Kommen die Daten von der PDP-11 oder einem Messgerät, welches die Adressrechnung im Gegensatz zur Siemens 330 von rechts nach links durchführt, so ist die zweite Form mit der Kennung B (Wort-Modus) zu wählen. Bei dieser Transferart werden die Bytes vor dem Schreiben oder nach dem Lesen vertauscht, damit sie sich der Adressrechnung der PDP-11 oder der Siemens 330 anpassen.

Liegt der Datensatz als Lochstreifen auf der Diskette, oder soll er in dieser Form auf die Diskette geschrieben werden, so ist die Transferart L zu wählen. Bei dieser Transferart wird kein Elementname (XXXXXX) angegeben, da sie nicht der Dateiorganisation von FLOARP entspricht. Der Datensatz wird Byteweise seriell bei Spur 0 Sektor 1 beginnend von der Diskette gelesen oder auf die Diskette geschrieben, bis das Ende durch 400 aufeinander folgenden Nullen erkannt wird. In dieser Form kann nur ein Datensatz pro Diskette umgesetzt werden.
Achtung: Keine eingerichtete und durch FLOARP belegte Diskette verwenden, da sie wegen des seriellen Zugriffs, durch das Überschreiben des Buchhalters zerstört würde.

Daten können nur in Blöcken mit der Größe eines ganzzahligen Vielfachen von 512 Bytes transferiert werden. Von der Platte (PLSK) kommende Datensätze, deren Länge nicht gleich einem ganzzahligen Vielfachen von 512 Bytes ist, werden entsprechend mit binären Nullen aufgefüllt.

3.7 ***** ENDE;etx *****

Mit diesem Kommando wird die Programmzuordnung für die Ebene 1 zurückgenommen und das Programm 'FLOARP' von "ablauffähig" in den Zustand "ruhend" gesetzt. Bei laufendem Transfer, kann mit der Anweisung '/ENDE:Pxxx;etx' -wobei xxx die Programmnummer angibt- das Programm beended und in Ruhestellung gesetzt werden. Für diese Beendigung wird vorausgesetzt, daß das Standardbedienprogramm eine höhere Priorität hat, als das zu beendende Programm (siehe ORG-Beschreibung 32.3.18).

4. FEHLERERKLÄRUNGEN

Grundsätzlich sind zwei verschiedene Fehlerarten zu unterscheiden. Zum ersten die, welche bei Aufrufen, die an das ORG gerichtet sind, auftreten, und zum zweiten die Fehler, die bei der Floppy-Verarbeitung auftreten können.

Bei Fehlern die in ORG-Aufrufen auftreten, wird FLOARP grundsätzlich abgebrochen und die virtuelle Adresse und der Inhalt des ersten Wortes des Aufrufparameterblocks (Detailierte Fehlerbitkennung siehe ORG-Beschreibung 40.12.) auf dem Blattschreiber aufgelistet.

V-ADR,	Parameterblocks,		ORG-Hinweis
1305	$SNEPEND		21.3.22
1309	$SNEPZU		21.3.20
1315	$SNEPRU		21.3.21
1321	$DATA		21.5.2
1327	$DAER		21.4.3
1331	$DAETL		21.4.10
1335	$DAEIER		21.4.2
1339	$dalv		21.4.9
1345	$DASL		21.4.6
1404	$STAUAL	(BDRA)	21.3.5
1412	$STEIAL	(BTAE)	21.3.5
1420	$BEDIEN		21.6.1
1428	$STAUAL	(DRUA)	21.3.5
1436	$STAUBI	(PLSK)	21.3.5
1444	$STEIBI	(PLSK)	21.3.5

Bei Fehlern, die während der Floppy-Verarbeitung auftreten, wird eine ausführliche Fehlermitteilung am Blattschreiber ausgegeben, und wenn nötig das FLOARP abgebrochen oder zur Eingabe-Routine verzweigt, damit der Benutzer die Arbeit wiederholen kann.

Treten an der Floppy-Disk Geräte-, Betriebs- oder EAP-Anzeigen auf, so werden diese vor Abbruch in hexadezimaler Form ausgegeben. Zusätzlich wird noch die Stelle, an der der Fehler aufgetreten ist, dem Benutzer durch dezimale Angabe von Laufwerk, Spur und Sektor bekanntgegeben.
Die genaue Fehlerbit-Information kann aus der Floppy-Disk-Beschreibung "E STE 4-129/000" entnommen werden.

5. RWFLOP

READ-WRITE-FLOPPY

Das RWFLOP ist ein Programmsegment, das als Erweiterung zum Floppy-Arbeits-Programm dem Anwender die Möglichkeit gibt, die Floppy-Disk im eigenen Programm so anzusprechen, wie es der Konvention von FLOARP entspricht. Er ist damit in der Lage, durch eigene Programm-Enwicklung zwischen jedem Gerät und der Floppy Daten zu transferieren, die die Floppy so belegen, wie es die Dateiorganisation von FLOARP verlangt.

RWFLOP besteht aus einem Einleitungsteil und zwei Unterprogrammen. Im Einleitungsteil werden die SNEP-Aufrufe ausgeführt, die Anschlußnummer der Floppy-Anschaltung übergeben und bestimmte Pointer für die Weiterverarbeitung gesetzt.
Das erste Unterprogramm, welches nur einmal durchlaufen wird, sorgt auf der Diskette für die Eröffnung des Elements, welches übertragen werden soll, und legt die Transferrichtung, den Übertragungsmodus und die zu übertragende Blockanzahl fest.
Mit dem zweiten Unterprogramm wird der Transfer durchgeführt. Es wird für jeden zu übertragenden Pufferinhalt einmal durchlaufen. Der Benutzer schreibt oder liest seine Daten aus dem bereitgestellten Puffer BLKPUF mit einer Größe von 512 Bytes, die er nicht überschreiten darf.

Der Anwender muß, um den reibungslosen Ablauf zu gewährleisten, folgende Punkte beachten.

1. Das RWFLOP muß an erster Stelle des Gesamtprogramms liegen, um die Abwicklung der Anlaufarbeiten und der SNEP-Aufrufe zu gewährleisten. Ist diese Anlaufsequenz durchlaufen, wird zu der ersten Anweisung im Anwenderprogramm verzweigt.

2. Die Parameter der Unterprogrammaufrufe sind immer zu setzen, bevor der Aufruf erfolgt (siehe unten).

3. Treten im Anwenderprogramm nach einer Elementeröffnung in der Floppy-Buchführung Fehler auf, so muß der Anwender in jedem Fall, wenn die Übertragung nicht zu Ende zu führen ist, das Unterprogramm FEHLER -(R0') :US FEHLER- anspringen, damit die bei der Anlaufsequenz gesetzten Pointer und der SNEP-Aufruf zurückgenommen werden können.

4. Alle Register stehen dem Anwender frei zur Verfügung außer Register 0. Dieses ist reserviert für die Adressenkellerung bei Unterprogramm-Sprüngen und kann zu diesem Zweck auch vom Anwender benutzt werden nach dem Schema:

```
(R0')   :US  SUBROUTIN ..... :SP  ('R0)
```

Der Aufruf für die Elementeröffnung wird mit der Angabe von drei hinter dem Aufruf liegenden Parametern durchgeführt.

(R0') :US OPENFL

Param. 1) Bit 0 Dieses Bit muß vor jedem Einsprung in das Unterprogramm auf 0 gesetzt sein. Tritt während der Floppy-Verarbeitung ein Fehler auf, so wird auf dem Blattschreiber eine Fehlermitteilung durch das RWFLOP erfolgen und dieses Bit auf eins gesetzt. Der Anwender muß nach jedem Aufruf dieses Bit abfragen und auswerten. Er verzweigt dann am besten zur Eingaberoutine und bestimmt hier durch entsprechende Eingabe, ob das Programm beendet oder mit erneuter Elementnamens-Angabe wiederholt werden soll.

Bit 1 =1 Übertragung erfolgt im Byte-Mode.
Bit 2 =1 Übertragung erfolgt im Wort-Mode.
Bit 3 =0 Gerät 0, =1 Gerät 1.
Bit 4 Byte-Adresse vom Elementnamen.
Bit 5 =0 Lesen, =1 Schreiben.

Param. 2) Adresse des Elementnamens (1-10 Zeichen).

Param. 3) Blockanzahl (512 Bytes je Block).
Ist dem Benutzer die Blockanzahl unbekannt, so muß er hier eine 0 eintragen. Dann wird die höchste zusammenhängende freie Blockanzahl der Diskette für den Transfer zur Verfügung gestellt und dem Benutzer durch Eintrag in diesem Parameter mitgeteilt.
Nach Beendigung des Transfers wird die benötigte Blocklänge als permanent eingetragen, und der bereitgestellte Rest wieder als Leereintrag gesetzt.

Der Aufruf für den Datentransfer wird mit einer Parameterangabe hinter dem Aufruf durchgeführt.

```
(R0')  :US  RWFLOP
```

Param. 1) Bit 0 Das Fehlerbit hat die gleiche Bedeutung wie oben.
Bit 15 Dieses Bit zeigt das Ende des Transfers an. Es wird im normalen Fall vom Benutzer gesetzt, wenn er die Subroutine anspringt, um den letzten Puffer zu übertragen. Daraufhin schließt RWFLOP nach der Übertragung dieses Puffers die Arbeit ab.

Bit 15 kann aber auch vom RWFLOP selbst gesetzt werden, und zwar dann, wenn RWFLOP die Arbeit von sich aus beendet, weil der letzte zur Verfügung stehende Block auf der Diskette beschrieben wurde.
Dieses ist gegeben, wenn der Datensatz größer ist als die Speicherkapazität der zu beschreibenden Diskette. Außerdem wird die Mitteilung "DISKETTE VOLL (WECHSELN)" über Blattschreiber ausgegeben.
Der Benutzer kann durch Abfragen dieses Bits dann bestimmen, ob auf einer zweiten Diskette weiter geschrieben werden soll. Er muß allerdings den zweiten Teil des Elements mit dem gleichen oder einem anderen Namen neu eröffnen und als erstes wieder durch das Unterprogramm OPENFL laufen.

BEISPIEL

Programmaufbau für die Benutzung des RWFLOP.
Diskette beschreiben

```
***          Programm RWFLOP        *** Muß ganz Vorne liegen
                •
                •
                •
***  EIN/    Texteingabe für Elementname oder Ende-Kommando
***          Bei Ende   :SP A5
                •
                •
                •
'IA'         G0 := 'H=2400'         *** Wort-Mode, Drive 0, Schreiben
'IA'         G1 := ELNAME           *** Adresse des Elementnamens
'IA'         G2 := 0                *** Blockanzahl ist unbekannt
'IA'         (PAR1) := G0           *** Parameter setzen
'IA'         (1+PAR1) := G1
'IA'         (2+PAR1) := G2
'IA'         (R0') :US OPENFL       *** Element eröffnen
'IA'  PAR1/  =0                     *** Parameter 1 Bitangaben
'IA'         =0                     *** Parameter 2 ADR.des Elementn.
'IA'         =0                     *** Blockanzahl oder 0
'IB'         (PAR1(0))=1 :SP EIN    *** Fehler im RWFLOP neue Eingabe
                •                   *** oder Arbeit beenden
                •
                •
'IA'  A1/    $RUFORG"DATEIN;        *** BLKPUF mit Daten füllen
***          Fehler ?  NEIN:  :SP A2
***          Einlesen wiederholen? JA:  :SP A1
'IA'         (R0') :US FEHLER       *** SNEP-Aufruf zurücknehmen
'IA'         $RUFORG"TEXTAUS;       *** Fehlertext ausgeben
'IA'         :SP EIN                *** Neustart oder beenden
***   A2/    Letzter Puffer? NEIN:  :SP A3
'IB'         (PAR2(15)) :T1         *** Endebit setzen
'IB'         (PAR2(14)) :T1         *** Elem.wurde vom Anwen.beendet
'IA'         :SP A4
'IA'  A3/    G0 := 0
'IA'         (PAR2) := G0
'IA'  A4/    (R0') :US RWFLOP       *** Daten auf Diskette schreiben
'IA'  PAR2/  =0
'IB'         (PAR2(0))=1 :SP EIN    *** Fehler im RWFLOP
'IB'         (PAR2(15))=0 :SP A1    *** Kein Ende,weiter einlesen.
'IB'         (PAR2(14))=0 :SP A6    *** Element wurde von RWFLOP
                                    *** beendet (Diskette voll).
'IA'         $RUFORG"TEXTAUS;       *** Text: Programmende.
'IA'  A5/    $RUFORG"STOP;
'IA'  A6/    $RUFORG"TEXTAUS;       *** Text: Diskette wechseln.
'IA'         :SP EIN                *** Auf einer neuen Diskette ein
                                    *** Element unter gleichem- oder
                                    *** neuem Namen eröffnen und
                                    *** weiter arbeiten
```

Gernot Zöller
Siemens AG - E STE 42

Karlsruhe, den 4.4.1978

Das Mikrocomputersystem 210

Inhaltsübersicht

Einleitung
Systemstruktur
Mikrocomputersystem 210 - Version D
Mikrocomputersystem 210 - Version E
Software
Anwendungen

Einleitung

Das Mikrocomputersystem 210 ist als kleines, aber wirkungsvolles Hilfsmittel zur Automatisierung konzipiert und für den industriellen Einsatz ausgelegt. Es umfaßt ein abgerundetes Spektrum von Flachbaugruppen, optionellen Bauelementen und den Zusatzeinrichtungen wie Testfeld, Busplatine und Stromversorgungsmoduln.

Das auf nationalen Normen und internationalen Empfehlungen beruhende universelle Einbausystem und die einfach gehaltene Systemschnittstelle gestatten es, mit minimalem Aufwand die verschiedenen Baugruppen in bestehende Geräte oder Systeme einzubinden bzw. das Mikrocomputersystem 210 durch spezielle, anwendungsspezifische Teile zu ergänzen. Der Einsatzbereich wird zusätzlich dadurch erweitert, daß die Flachbaugruppen im einfach und doppelt hohen Europaformat zur Verfügung stehen. (Version E und Version D.)

Speziell für den Einsatz bei kleinen Automatisierungsaufgaben ist die feine Stufung im Ausbau des wiederprogrammierbaren Festwertspeichers von 1 KBytes und die des Schreib-Lese-Speichers von 4 KBytes vorgesehen. Die Flexibilität des Systems 210 zeigt sich auch in der mischbaren, platzunabhängigen Bestückung der Busplatine, die in ihrer Länge dem Anwendungsfall angepaßt wird und die Systemkomponenten miteinander verbindet.

Für kleinste Anwendungen ist insbesondere der Mikrocomputer 210 D attraktiv, da er auf nur einer doppelt hohen Flachbaugruppe bereits einen kompletten Rechner (single-board-computer) inclusive Speicher und Interface darstellt.

Durch die Verwendung der bewährten und weltweit eingeführten Bausteinfamilie mit dem Mikroprozessor 8080 ist die Weiterentwicklung des Mikrocomputersystems 210 sichergestellt.

Für die Entwicklung der Software steht der Programmerstellungs- und Testplatz SME-800 in verschiedenen Komfortstufen zur Verfügung. Die Programmerstellung kann jedoch auch auf Cross-Rechnern wie z. B. den Siemens Systemen 7000 erfolgen.

Systemstruktur

Das Mikrocomputersystem 210 ist ein standardisiertes Baugruppenspektrum. Es ermöglicht dem Anwender, aus fertigen Moduln Prozeßrechnerfunktionen für den untersten Automatisierungsbereich einfach zusammenzustellen. Für die problemlose Integration in die verschiedenen zu automatisierenden Geräte steht das System 210 sowohl im einfach wie im doppelt hohen Europaformat des Einbausystems 902 mit den Abmessungen nach DIN 41494 und Steckern nach DIN 41612 zur Verfügung. Beide Versionen sind mit derselben busfähigen Systemschnittstelle ausgerüstet und können daher auch gemischt verwendet werden.

Für solche Anwendungen, wo die im Mikrocomputer 210 bereits integrierten Funktionen nicht ausreichen, stehen zur Erweiterung Speicher- und Ein-/Ausgabemoduln, steckbare Bauelemente und eine Busplatine für die mechanische und elektrische Verbindung zur Auswahl. Das Baugruppenspektrum wird durch ein Testfeld und Stromversorgungsmoduln abgerundet, mit denen auch der eigenständige dezentrale Einsatz in einer konstruktiv geschlossenen Form - z. B. in einem Baugruppenträger des Einbausystems 902 - möglich ist.

Der Mikrocomputer 210 bietet bereits im Minimalausbau eine Bearbeitung von 8 Interrupts, einen Ein-/Ausgabeverkehr wahlweise nach dem Speicherseiten- oder dem isolierten Ein-/Ausgabeverfahren und eine Adressierung des maximalen Speicherausbaus von 64 KBytes über 16 Bit breite Adressen. Für die Informationsverarbeitung stehen 78 Befehle, 1 Akku, 6 Mehrzweck- und 2 Spezialregister zur Verfügung.

Mikrocomputersystem 210 - Version D

Wo Computerleistung im doppelt hohen Europaformat benötigt wird, bietet der Mikrocomputer 210 D als single-board-computer die kompakteste Lösung:
bereits auf einer Flachbaugruppe mit den Abmessungen 233,4 mm x 160 mm sind der Prozessor mit Prioritätssteuerung, bis zu 4 KBytes wiederprogrammierbarer Festwertspeicher, 1 KBytes Schreib-Lese-Speicher, ein parametrierbarer Zeitgeber und ein Interface mit umschaltbarer Schnittstelle nach DIN 66020 (V.24) oder für 20 mA Linienstrom (TTY) realisiert.

Wenn bei komplexeren Anwendungen 5 KBytes Speicher für Programme und Daten nicht ausreichen, können verschiedene Speichermoduln mit bis zu 12 KBytes auf einer Flachbaugruppe nachgerüstet werden. Um optimal angepaßte Lösungen zu erreichen, ist der Speicherausbau in Stufen von 1 KBytes EPROM oder 4 KBytes CMOS-RAM möglich.

Entscheidenden Einfluß auf den wirtschaftlichen Einsatz in den verschiedensten Geräten und Systemen hat das umfangreiche Spektrum der Signalformer. Hier stehen in unterschiedlichen Ausführungen bezüglich der Leistungsdaten und Anschlußbedingungen Flachbaugruppen für Eingaben und Ausgaben von digitalen und analogen Signalen zur Auswahl.

Für spezielle Anforderungen wird das vorliegende Spektrum durch zusätzliche Ausführungen noch erweitert. Die einfache Systemschnittstelle und die steckplatzunabhängige Verbindung über eine in der Länge variable Busplatine bietet darüber hinaus die besten Voraussetzungen für kundenseitige Spezialentwicklungen.

Mikrocomputersystem 210 - Version E

Für zahlreiche Anwendungsfälle werden auch Flachbaugruppen im einfach hohen Europaformat benötigt. Unter Beibehaltung der bewährten Struktur, derselben Systemschnittstelle und der gleichen Flexibilität in der Leistung und Ausbaufähigkeit wurden darum die Funktionseinheiten des

Mikrocomputersystems 210 auf Flachbaugruppen mit den Abmessungen 100 mm x 160 mm realsiert. Ein Mikrocomputer 210 E kann z. B. aus drei derartigen Moduln mit demselben Funktionsumfang wie der Mikrocomputer 210 D projektiert werden.

Entsprechend umfangreich ist das Spektrum an Speichermoduln mit maximal 8 KBytes auf einer Flachbaugruppe, mit denen der Speicher bis zum Maximalausbau von 64 KBytes in Stufen von 1 KBytes EPROM oder 4 KBytes NMOS-RAM erweitert werden kann.

Von besonderer Bedeutung für die wirtschaftliche Integration des Systems 210 in die zu automatisierenden Geräte und Systeme ist der einfache Anschluß von Peripheriegeräten und die Kopplung zu anderen Rechnern. Dies können z. B. Mikrocomputer in einem Mehr-Prozessor-System oder Minicomputer und Prozeßrechner der Siemens Systeme 300 sein.
Hierfür existiert auf einer eigenen Flachbaugruppe ein Interface, das wie im Mikrocomputer 210 D eine umschaltbare, serielle Schnittstelle nach DIN 66020 oder für 20 mA Linienstrom und gleichzeitig einen parametrierbaren Zeitgeber mit drei 16-Bit-Registern bietet.

Software

Die Wirtschaftlichkeit des Einsatzes von Mikrocomputern steigt sehr stark mit der Zahl identischer Anwendungen, da sich die Softwareerstellungskosten entsprechend verteilen und sich dem Anteil für den Kopiervorgang - den Kosten für das Laden des wiederprogrammierbaren Festwertspeichers - nähern.

Bei den erzielbaren niedrigen Hardwarekosten eines Mikrocomputersystems kommen der rationellen Softwareerstellung und der Möglichkeit, entsprechend dem Anwendungsfall laufzeit- oder speicherplatzoptimale Programme erzeugen zu können, um so stärkeres Gewicht zu.

Neben der bei Kleinstanwendungen in Frage kommenden Programmierung in der Maschinensprache steht beim Mikrocomputersystem 210 für eine rationelle Softwareerstellung ein Siemens-Mikrocomputer-Entwicklungssystem SME-800 in der gewählten Komfortstufe zur Verfügung.

Die Assemblersprache SAB 8080 und die höhere Programmiersprache PL/M stehen zur Verfügung. Der Programmtest wird durch Echtzeitemulation auf einem SME-800 durchgeführt.

Durch die Wahl der Hardwareausrüstung und den Umfang der Softwarehilfsmittel kann das SME im Komfort in einem sehr weiten Bereich der Anforderungen an die Softwareerstellung und der zu erwartenden Komplexität der Software angepaßt werden. Der Funktionsumfang der Erstellungshilfen reicht von der einfachen Programmerstellung und -archivierung durch Lochstreifen bis zum komfortablen Erstellung modularer, speicherplatzunabhängiger Programmstrukturen über Sichtgerät, Floppy-disk bzw. Text-Editor, Binder, Lader und Betriebssystem. Entsprechend vollständig wurden die Funktionen für die Durchführung und Protokollierung des Echtzeittests realisiert.

Die Erfahrungen aus den bisher durchgeführten Anwendungen des Mikrocomputersystems 210 stehen dem Softwareentwickler in Form von Software-Treiberroutinen z. B. für die Kupplung, für das Ansprechen von Peripheriegeräten oder Signalformern und weitere häufig auftretende Routinen als Programmoduln und Protokolle zur Verfügung.

Anwendungen

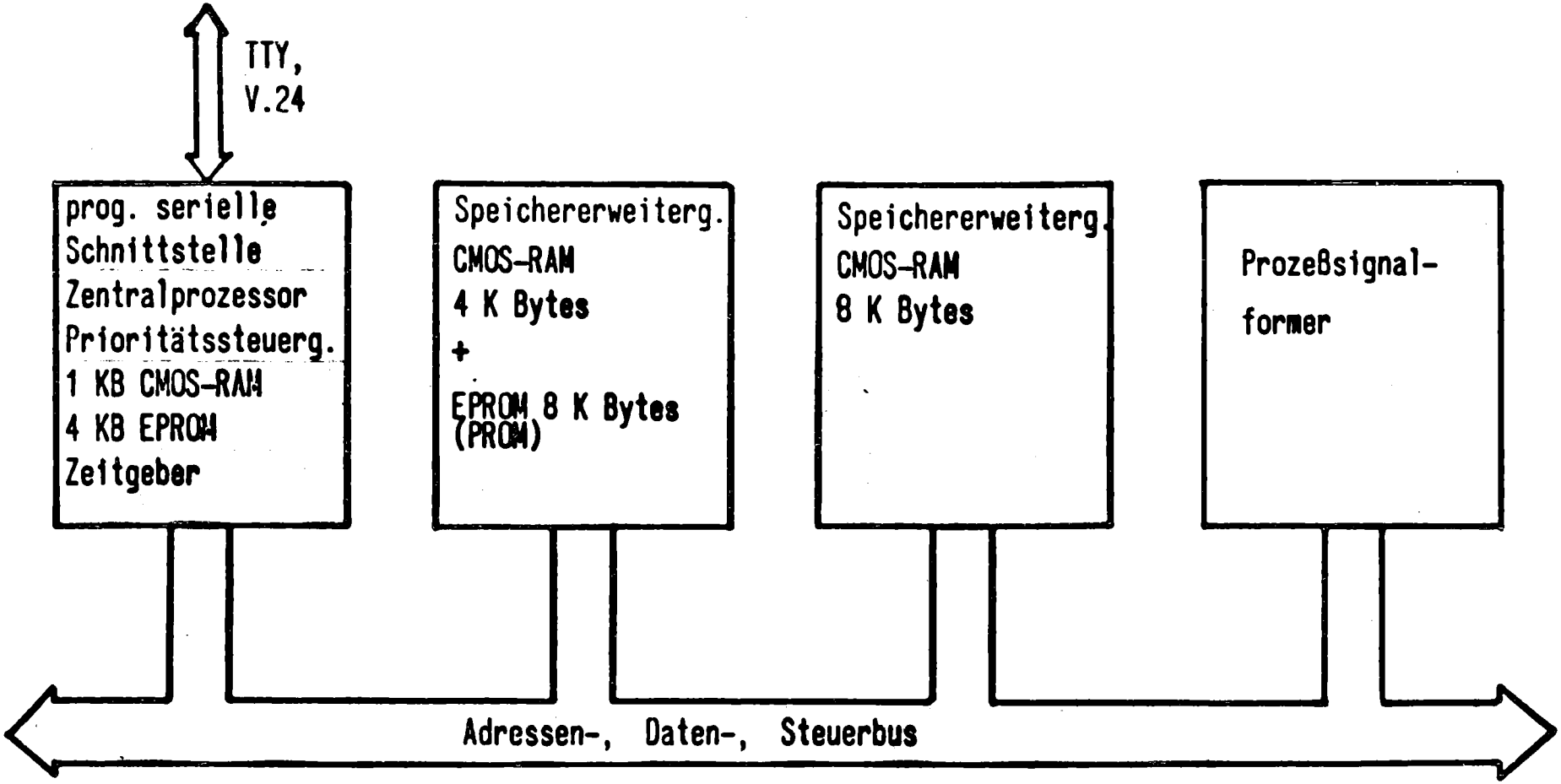
Als abgerundetes Flachbaugruppenspektrum im industriellen Aufbau, mit genormten Abmessungen in zwei gemischt projektierbaren Versionen und mit seiner modularen Aufbaufähigkeit bietet es Standardisierung, Flexibilität und wirtschaftliche Integration zugleich.
Das System 210 ist vom Anwender einfach zu erweitern und problemlos mit weiteren Systemen 210 oder leistungsfähigen Rechnern, z. B. den Prozeßrechnern der Siemens Systeme 300, zu koppeln.

Die freie Programmierbarkeit und die Möglichkeit, die Programmiersprache und den Programmierkomfort entsprechend dem geplanten Entwicklungsumfang bestimmen zu können, ermöglichen einen wirtschaftlichen Einsatz des Mikrocomputersystems 210 schon bei geringen Stückzahlen.

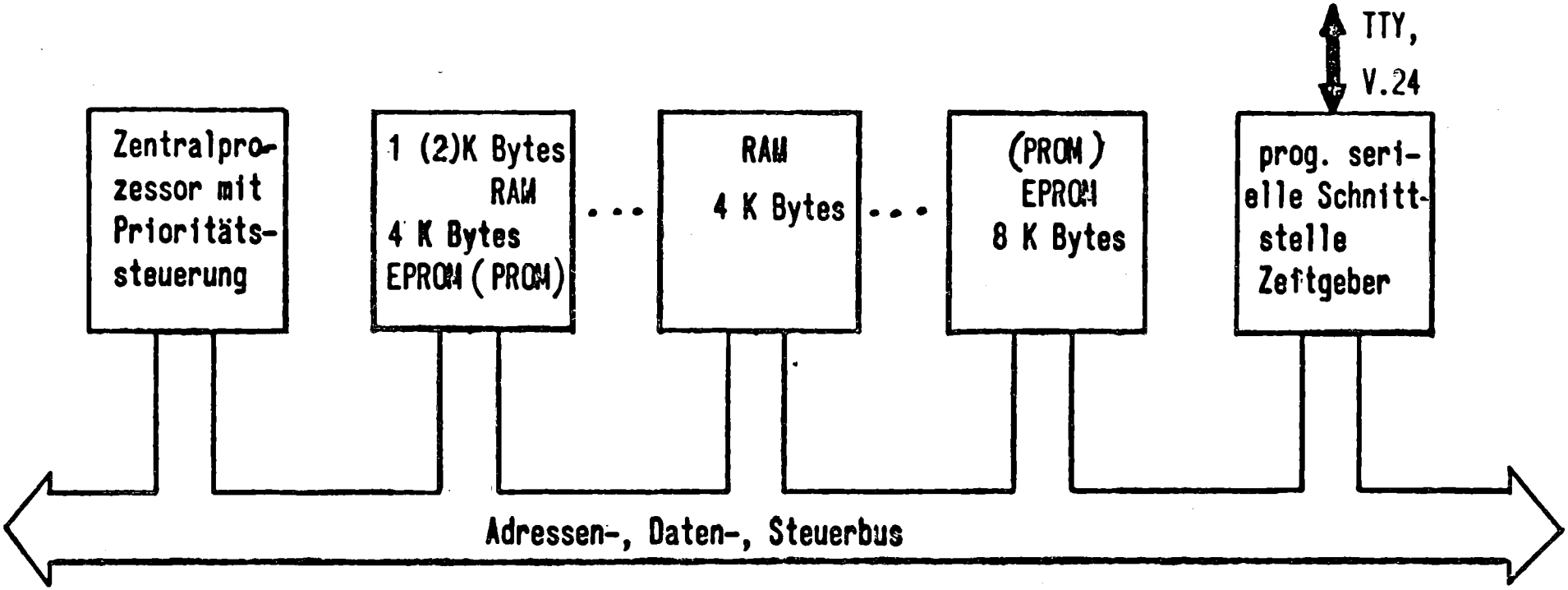
Die Siemens-internen Anwendungen sind bisher:
- Ablösung komplexer, verbindungsprogrammierter Logik (S31-210)
- Industrielle Steuerungen (CEPAMAT, PRINTAMAT, Papier- und Druckindustrie)
- Meßdatenerfassung (Meldedrucker MD 31)
- Fernwirksysteme (SINAUT 8FW)

Literaturhinweise:

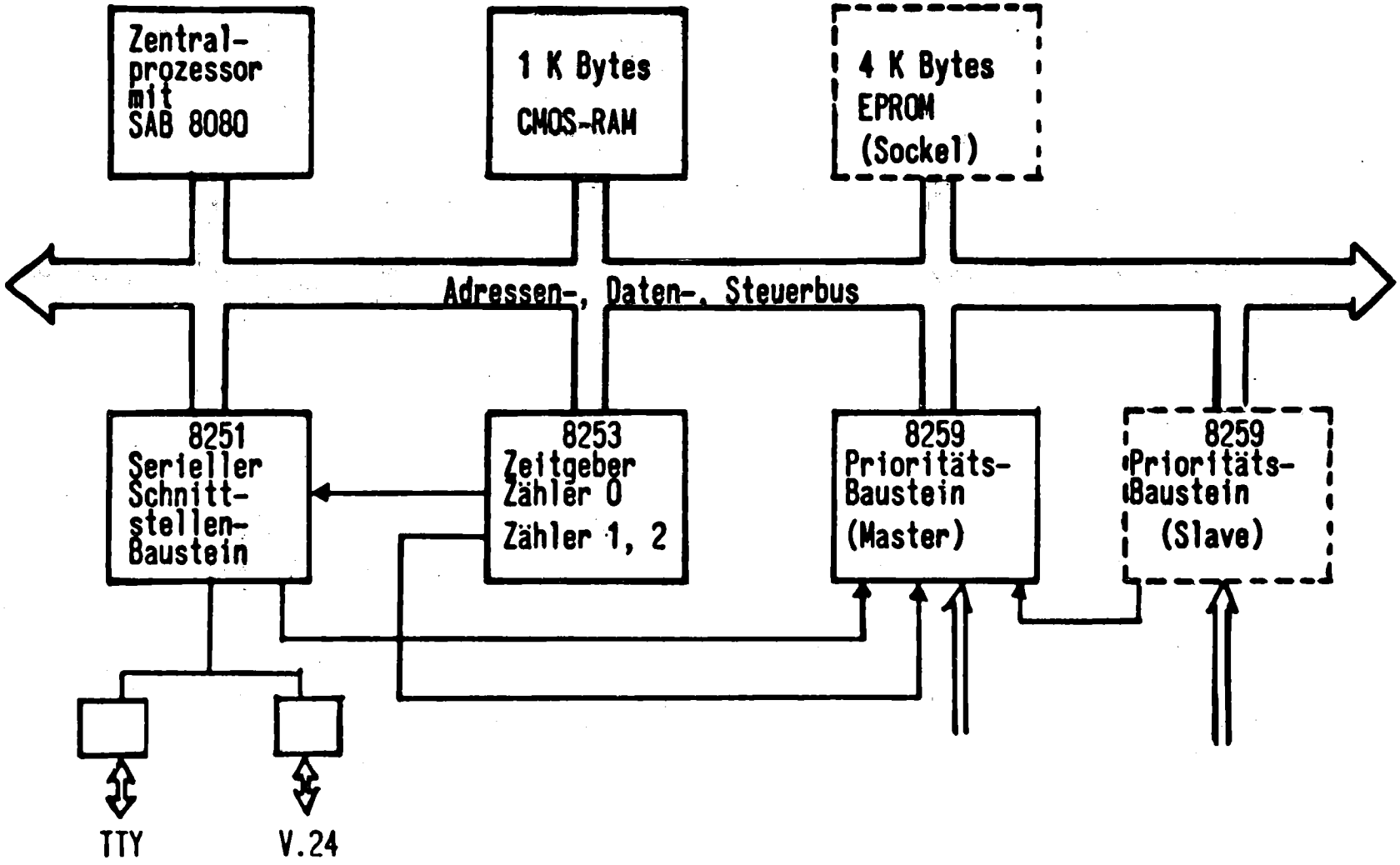
Mikrocomputersystem 210 - Systembeschreibung	A112/2222
Mikrocomputer 210 E	A112/2257
Mikrocomputer 210 D	A112/2225
Mikrocomputersystem 210 - Speichermoduln für Version E und D	A112/2288
Mikrocomputersystem 210 Ein/Ausgabemoduln	A112/2289



Mikrocomputer-system 210 D : Funktionseinheiten auf Flachbaugruppen von doppeltem Europaformat des ES 902
(1 Kästchen ≙ 1 FBG mit den Abmessungen
160 mm x 233,4 mm)



Mikrocomputer-system 210 E : Funktionseinheiten auf Flachbaugruppen von einfachem Europaformat des Einbausystems 902
(1 Kästchen ≙ 1 FBG mit den Abmessungen 160 mm × 100 mm)



Funktionsübersicht des MC 210

Standard-Signalformer für das Mikrocomputersystem 210

Baugruppe

---

stat. DE

- 32 Eing., potentialgetrennt
- Eing.-Spannung: 24/48÷60 V

---

stat. DA elektronisch

- 32 Ausg., potentialgetrennt
- Ausg.-Spannung TTL/24V/max.45V

---

stat. DA mit Haftrelais

- 32 Ausg., potentialgetrennt
- Ausg.-Spannung: max. 60 V

---

integr. AE 11 Bits + Vz,freilaufend

- max. 32 Eingänge (mit 2 MW)
- Eing.-Spannung: projektierbar (± 0,3 bis ± 10V), ± 1V, ± 10V
- Eing.-Strom : projektierbar (± 1 bis ± 50 mA), ± 10 mA, ± 20 mA

Meßstellenwähler 2-polig

- 16 Eing., potentialgetrennt durch Relais

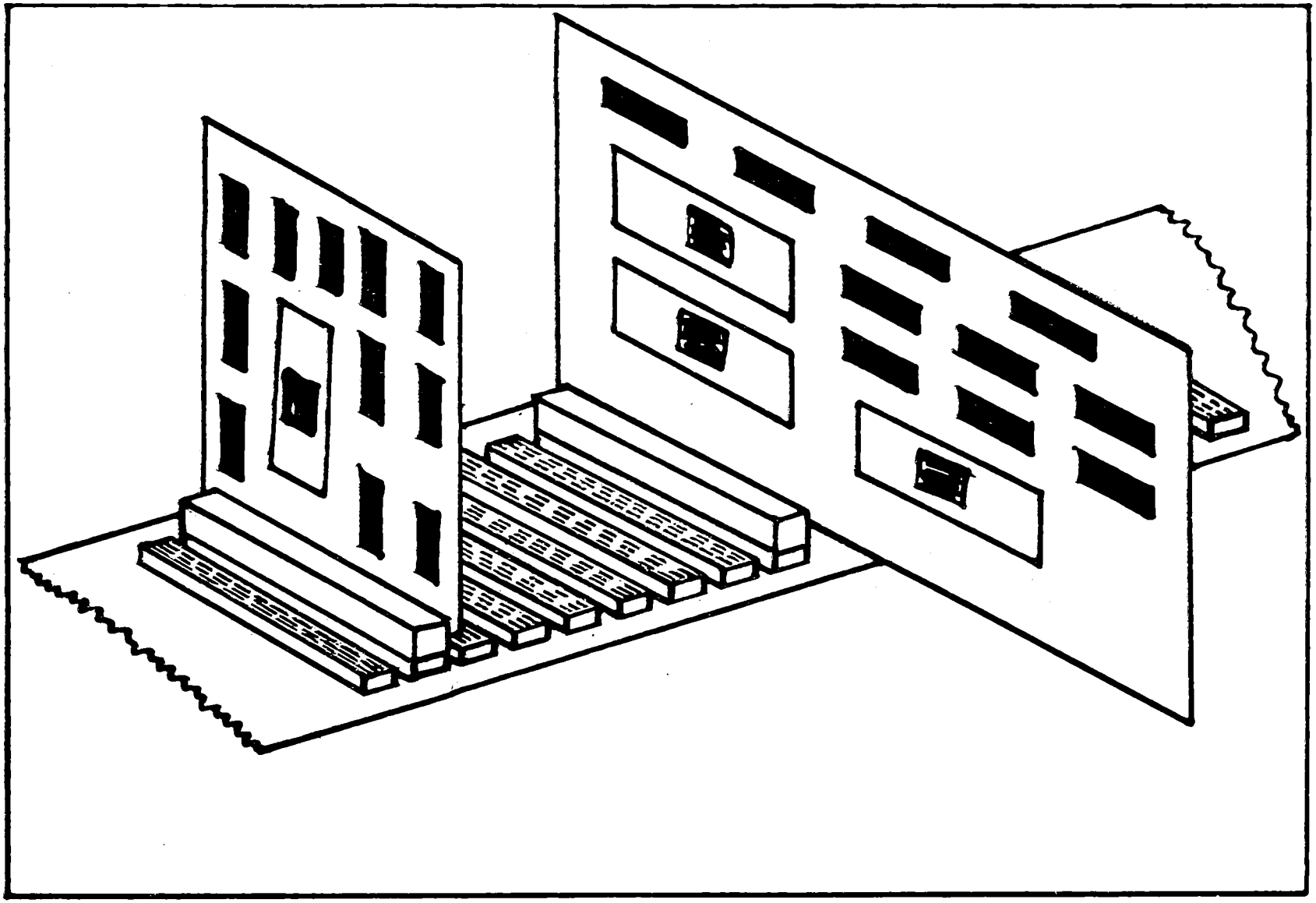
---

integr. AE 8 Bits + Vz, freilaufend

- 32 Eing., potentialgebunden
- Eing. -Spannung: ± 1V (± 2V)
- Eing.-Strom : projektierbar (± 1 bis ± 50 mA), ± 10 mA, ± 20mA

Analogausgabe 8 Bits + Vz

- 6 Ausgänge, potentialgebunden
- Ausg.-Strom: projektierbar (± 2,5 bis ± 20 mA), ± 10 mA, ± 20 mA
- Ausg.-Spannung:±7 V



Mikrocomputersystem 210
Busplatine mit Flachbaugruppen des Systems 210 E und 210 D

Josef Rinder
SIEMENS AG - E STE 42

Karlsruhe, den 6.4.1978

Neue Komponenten der SIEMENS-Systeme 300-16 Bit

Inhaltsverzeichnis:

Modellreihe 330-R10, R20, R30
. Zielsetzung
. Technische Daten
. Aufbautechnik
Neues Anschaltungskonzept
Externspeicher 3947, 3948, 3958
Prozeßsichtgerät 3977

Die Modellreihe 330-R10, R20, R30

Die Minicomputer 330-R10, 330-R20 und 330-R30 erweitern die SIEMENS-Systeme 300 um eine Modellreihe, die sich auf bewährte Systemfunktionen abstützt, jedoch durch fortschrittliche innovative Technik besticht.

Zusammen mit dem Spitzenmodell, dem Multiprozessorsystem 340-R40, bieten die Systeme R10, R20 und R30 ausgeprägte funktionelle Gemeinsamkeiten. Diese ermöglichen es, aufbauend auf einer einheitlichen Grundsoftware, eine kompatible Anwendungssoftware bereitzustellen.

Zielsetzung

Mit der Entwicklung dieser Zentraleinheiten sind keine neuen Systeme geschaffen worden, wie es z. B. beim Übergang von den Systemen 300-24 Bit zu den Systemen 300-16 Bit notwendig war, sondern aufbauend auf den Erfahrungen mit der vorangegangenen Modellreihe wurde die Leistungsabgrenzung neu konzipiert.

Im einzelnen wurden folgende Ziele verfolgt:

. Verbesserung des Preis/Leistungs-Verhältnisses
  (d.h. Preisreduzierungen und/oder Leistungserhöhung)

. Übernahme der für den Prozeßrechner 330 existierenden
  Anwendungssoftware ohne besonderen Anpassungsaufwand
  (keine Kosten, großer Umfang, erprobt)

. Möglichst geringen Änderungsaufwand an der für die Zentraleinheit 330 existierenden Grundsoftware:
  - Organisationsprogramm (ORG 330 K)
  - Übersetzerprogramme
  - Dienst- und Hilfsprogramme

. Kein Änderungsaufwand zur Übernahme der Systemperipherie

. Bestmögliche Leistungsabstufung der einzelnen Modelle mit sinnvoller Modularität, d. h. die Unterschiede möglichst nur in quantitativen Eigenschaften wie Befehlsausführungszeiten oder Zentralspeicherausbau.

Technische Daten

Die Gemeinsamkeiten werden deutlich, wenn wir die Struktur der innovierten Zentraleinheiten betrachten (Anlage 1):

Sie bestehen aus dem zentralen, mikroprogrammierten Steuer- und Rechenwerk für

. die Befehlsliste,
. die Unterbrechungsstruktur und
. die Grundstufe des Ein-Ausgabe-Systems (Integrierter Ein-Ausgabe-Prozessor IEAP)

Diese Grundausstattung kann durch

. einen Gleitpunktprozessor und
. Selektor-Ein-Ausgabe-Prozessoren (SEAP)

erweitert werden. Gleichzeitig wird schon aus dem Aufbau die Abgrenzung zu dem Multiprozessorsystem 340-R40 deutlich. Die Leistungsunterschiede im einzelnen gehen aus der "Technischen Übersicht" (Anlage 2) hervor.

Aufbautechnik

Die Anlagen 3 bis 6 zeigen den Aufbau der Zentraleinheiten im Rahmen.

- Die ZE 330-R10: Zentralprozessor, IEAP, ZSP, GP und 6 EA-Anschlußstellen mit Stromversorgung in einem 19-Zoll-Baugruppenträger des SIEMENS-Einbausystems 902.

- Die ZE 330-R20: Zentralprozessor, IEAP, ZSP, GP und 12 EA-Anschlußstellen in einem 19-Zoll-Baugruppenträger des SIEMENS-Einbausystems 902. SV im getrennten Baugruppenträger.

- Die ZE 330-R30: Zentralprozessor, IEAP, GP, 12 EA-Anschlußstellen und ZSP-Anpassung in einem 19-Zoll-Baugruppenträger des SIEMENS-Einbausystems 902.
  ZSP und SV in jeweils separaten Baugruppenträgern.

Neues Anschaltungskonzept

Im Betriebssystem ORG 300 für die neue Modellreihe wird für ähnliche Geräte auch ein gemeinsamer universeller Gerätetreiber eingesetzt. Damit kann auch das schon seit einiger Zeit verfolgte Anschaltungskonzept optimal genutzt werden. D. h. mit einer Grundbaugruppe mit Mikroprozessor und serieller Schnittstelle werden z. B. Drucker und Sichtgeräte an die Ein-Ausgabe-Anschlußstelle der SIEMENS-Systeme 300 angepaßt, wobei die spezifischen Geräteeigenschaften mittels Firmware auf den universellen Gerätetreiber abgestimmt werden.

Bei aufwendigeren Steuerfunktionen der peripheren Geräte und/ oder parallelen Schnittstellen wird die Grundbaugruppe durch

eine zweite Flachbaugruppe ergänzt. Die Anlage 7 zeigt das prinzipielle Konzept.

Externspeicher 3947, 3948, 3958

Analog zur Innovation der Zentraleinheiten aufgrund neuer Technologien wird auch das Spektrum der Peripheriegeräte laufend ergänzt bzw. durch leistungsgerechtere Einheiten ersetzt. So bietet sich im Bereich der Plattenspeicherlaufwerke durch größere Bitdichten, höhere Datenraten und/oder kürzere Zugriffszeiten eine Ergänzung des Spektrums der bewährten Laufwerke an. Für den Anschluß an die Rechner der neuen Modellreihe steht ein Selektor-Ein-Ausgabe-Prozessor zur Verfügung, der die Ausnutzung der hohen Datenraten durch direkten Zentralspeicherzugriff ermöglicht.

. Plattenspeichereinheit 3947
  - Steuerung mit max. 4 Laufwerken.
  - Fest eingebaute Magnetplatte in einem staubdicht abgekapselten Gehäuse.
  - Mittlere Zugriffszeit 50 ms
  - Max. Netto-Speicherkapazität 11,4 MByte pro Laufwerk.
  - Mittlere Datenrate 820 kBytes/sec.
  - Festsektoriertes Format mit 512 Bytes/Sektor.
  - Zugriff zu logischen Sätzen beliebiger Länge durch automatische Sektor- und Zylinderfortschaltung.
  - Datenpuffer von 1 kBytes zur Entkopplung der Zeitbedingungen zwischen Zentraleinheit und Laufwerk sowie zum byteweisen Zugriff innerhalb eines Sektors.

. Plattenspeichereinheit 3948
- Plattenstapel mit 5 Platten, wovon 9 Oberflächen zur Speicherung der Nutzinformation dienen.
- Mittlere Zugriffszeit 35 ms.
- Maximale Speicherkapazität (netto) 120 MBytes pro Laufwerk.
- Mittlere Datenrate 655 kBytes/sec.
- Format, Zugriff und Steuerung wie bei 3947.

. Magnetbandeinheit 3958
- Steuerung mit max. 4 Laufwerken
- Pneumatische Bandzugsteuerung für datenträgerschonende schnelle Bandoperationen.
- Bandgeschwindigkeit 50 ips bei einer wählbaren Aufzeichnungsdichte von 800 bpi oder 1 600 bpi
- Aufzeichnung auf 1/2-Zoll-Magnetband mit 9-Spur-Format.
- Datei- und Etikettaufbau lehnen sich weitgehend an DIN 66 029 an. Standardmäßige Umcodierung zwischen ISO-7-Bit-Code und EBCDI-Code.
- Max. Speicherkapazität (netto) 20 - 35 MBytes
- Mittlere Datenrate 40 bzw. 80 kBytes/sec.
- Variable Blocklänge
- Selbständige Fehlerkorrektur
- Datenpuffer von 2 kBytes zur Entkopplung der Zeitbedingungen zwischen ZE und Laufwerk.

Prozeßsichtgerät 3977

Das Prozeßsichtgerätesystem 3977 dient zur Ein- und Ausgabe von alphanumerischen Texten, graphischen Darstellungen mittels Symbolen, punktweisen Kurven und digitalen Signalen. Es besteht aus einer intelligenten Steuerung und den Ein- und Ausgabegeräten. Anlage 8 zeigt den Umfang sowie die Funktions-Erweiterungen gegenüber der Graphik-Bildschirmeinheit 3976.

Zentraleinheiten 330-R10, R20, R30, 340-R40 Ausführung

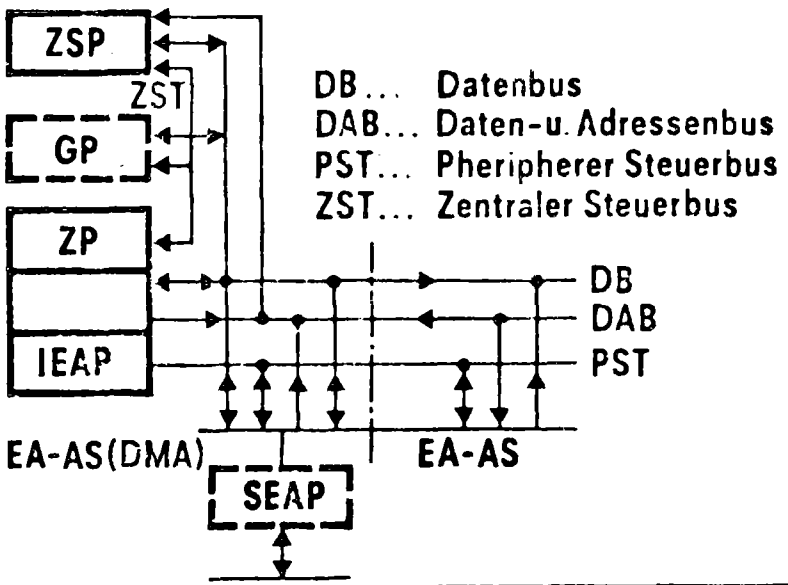
ZE 330-R10, R20, R30

Zentrales, mikroprogrammiertes Steuer-und Rechenwerk für:

- Befehlsliste
- Unterbrechungsstruktur
- Grundstufe des Ein-Ausgabe-Systems (Integrierter Ein-Ausgabe-Prozessor)

modular erweiterungsfähig durch

- Gleitpunktprozessor
- Selektor-Ein-Ausgabe-Prozessoren



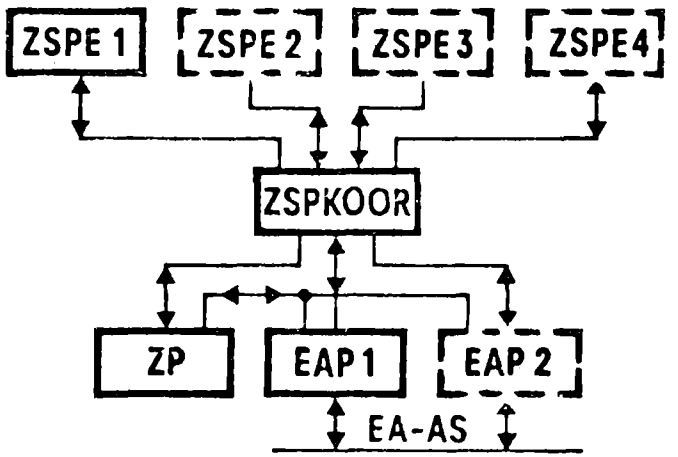
ZE 340-R40

Multiprozessorsystem mit autonomen Prozessoren für

- Zentralprozessor
- EA-Prozessor (max.2)

und unabhängigen Zentralspeicherelementen (max.4)

ZP...	Zentralprozessor
EAP1,2...	Ein-Ausgabe-Prozessor 1,2
ZSPKOOR...	Zentralspeicherkoordinator
ZSPE1,2,3,4...	Zentralspeicherelement 1,2,3,4



Siemens Systeme 300

Struktur der innovierten Zentraleinheiten

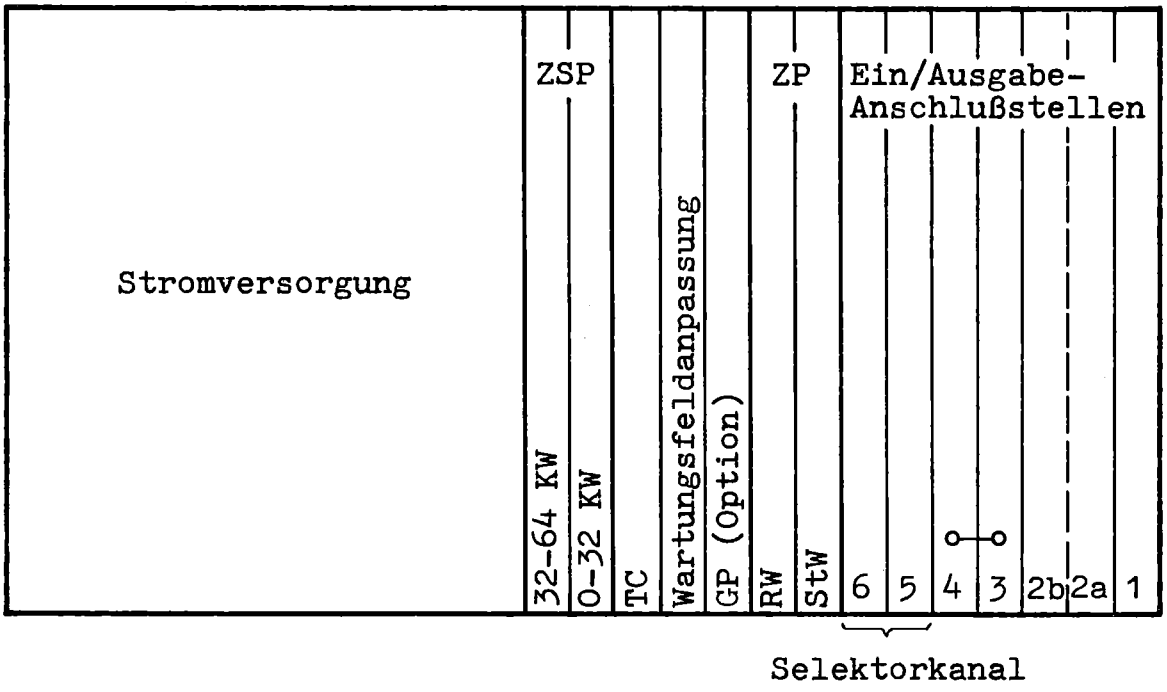
EESTE41
F771371

Zentraleinheiten	330-R10	330-R20	330-R30	340-R40
Standardregister	16 im Zentralspeicher	16 im Zentralprozessor (schneller TTL-Registersatz)		
Parallelverarbeitung	16 Bits (32 Bits bei GP-Arithmetik)			16, 32, 64 Bits
Befehle (Anzahl)	234 Standardbefehle (+ 85 zusätzliche Befehle für 32-, 64-Bit-GP- und 32-Bit-FP-Arithmetik)			323
Operationszeit (RR) [µs]	2,8	0,9		0,65
Unterbrechungssystem	16 Zustände, 2 Modi			6 Zustände
Adressierung	reell		virtuell	
Adressiervolumen für Programme	$GRP + CB + \Sigma AWP_i \leq 64$ kW		$CB + AWP_i \leq 64, 128$ kW; $GRP + \Sigma CB + \Sigma AWP_i \leq 256$ kW	≤ 512 kW
EA-Anschlußstellen	6	12		2 × 8
Ein-Ausgabe-Prozessor (EAP)	Grundausbau: integrierter EAP (IEAP) max. 2 SEAP	max. 4 Selektor-EAP (SEAP)		1 oder 2 EAP
Transfergeschwindigkeit [kW/s] (jeweils max.)	an 1 EA-Anschlußstelle: IEAP: 400 SEAP: 600	IEAP: 400 SEAP: 700		1 EAP: 1000
	über mehrere EA-Anschlußstellen: IEAP: 200 SEAP: 1200	IEAP: 200 SEAP: 1400		2 EAP: 2000
Speicherart	Halbleiterspeicher (dyn. RAM)			Kernspeicher
Zugriffs-/Zykluszeit [ns]	300/450			400/1000
Ausbau/Stufen [kW]	64/16, 32		256/16, 32	512/64
Schaltkreistechnik	(Low Power-Schottky-) TTL in MSI und LSI; N-MOS in LSI			
Einbausystem	Siemens ES 902 nach DIN und IEC; 19"-Baugruppenträger; Abmessungen der FBG: 233 mm × 160 mm; 48- und 64poliger Steckverbinder			
Aufbau	Baugruppenträger (BGT): 1 BGT inkl. Stromversorg.	1 BGT	2 BGT	min. 4 BGT
	Schrank: Abmessungen (B×H×T) 800×1800×700 mm³			
	Bildschirmarbeitsplatz SICOMP R10	SICOMP R20		

Siemens Systeme 300
Technische Übersicht innovierte Zentraleinheiten

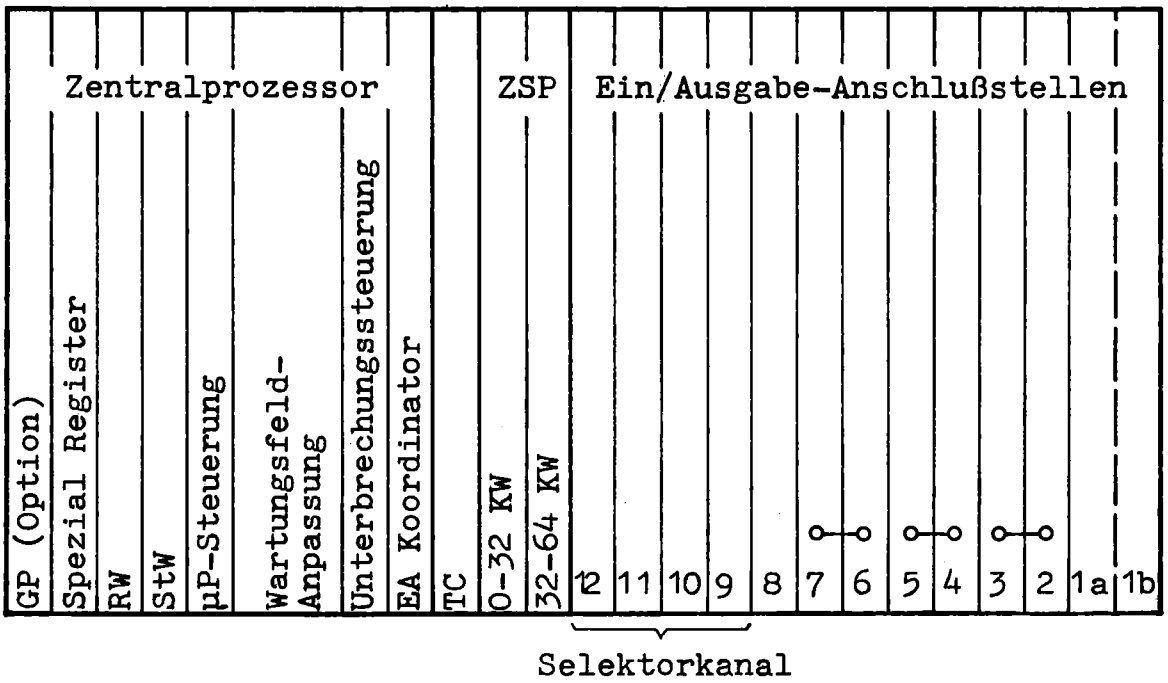
EESTE F771381

Anlage 3



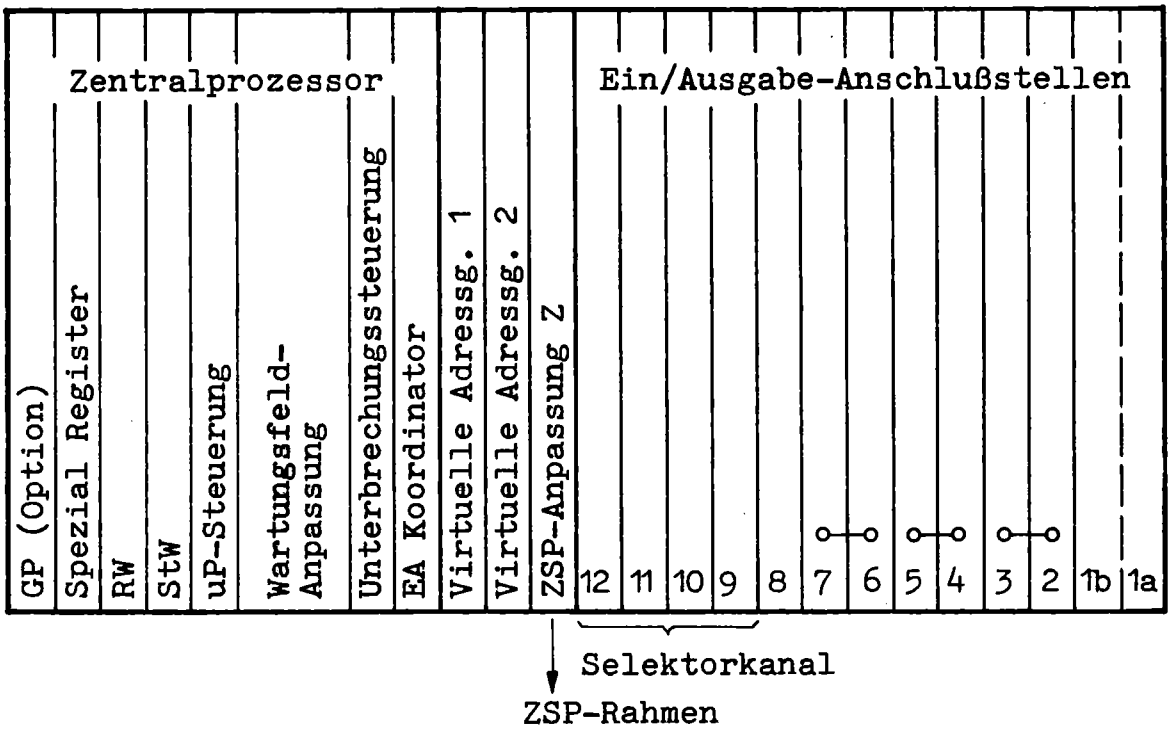
ZE 330-R10 Rahmenaufteilung

Anlage 4



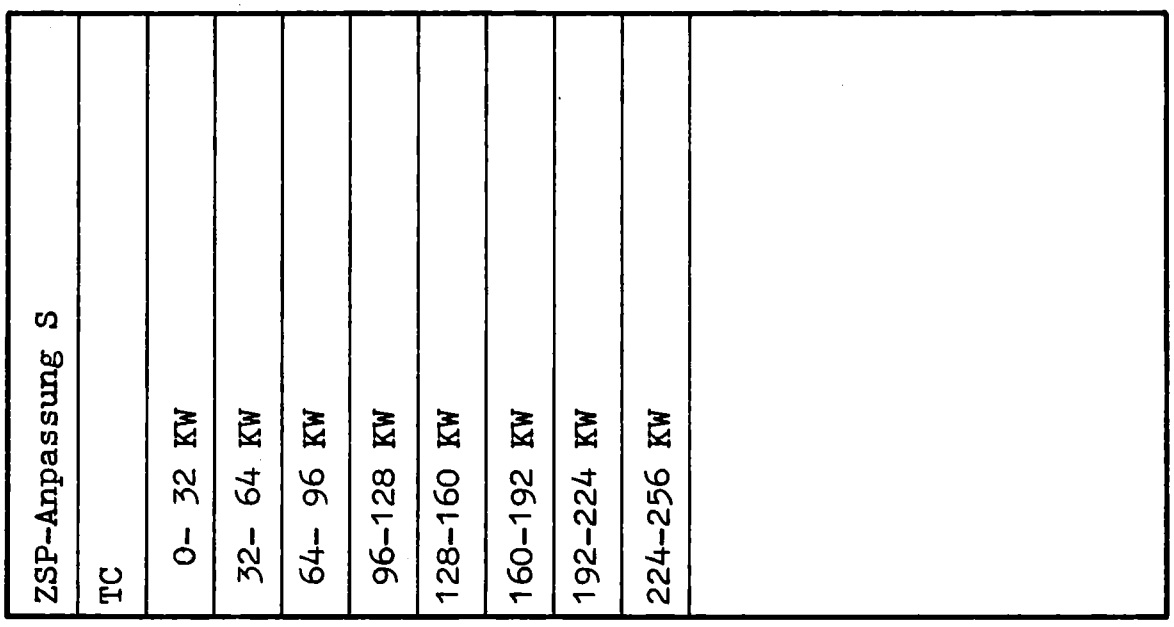
ZE 330-R20 Rahmenaufteilung

Anlage 5

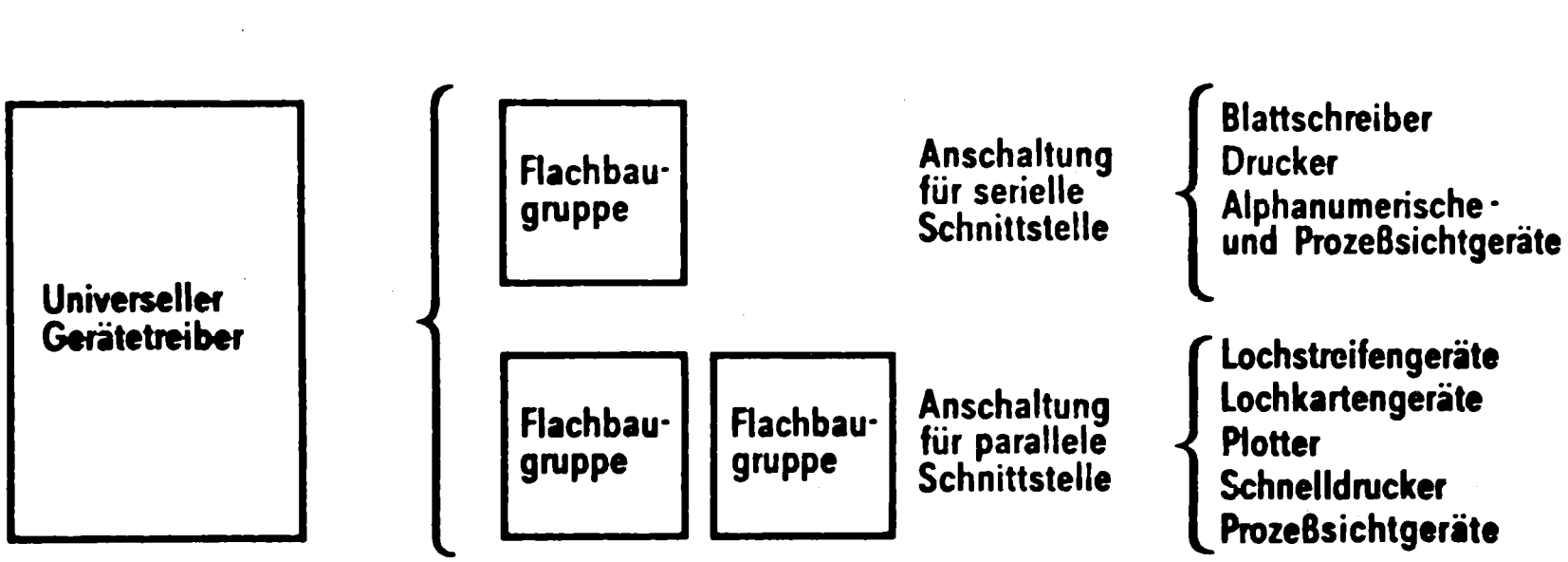


ZE 330-R30 Rahmenaufteilung

Anlage 6



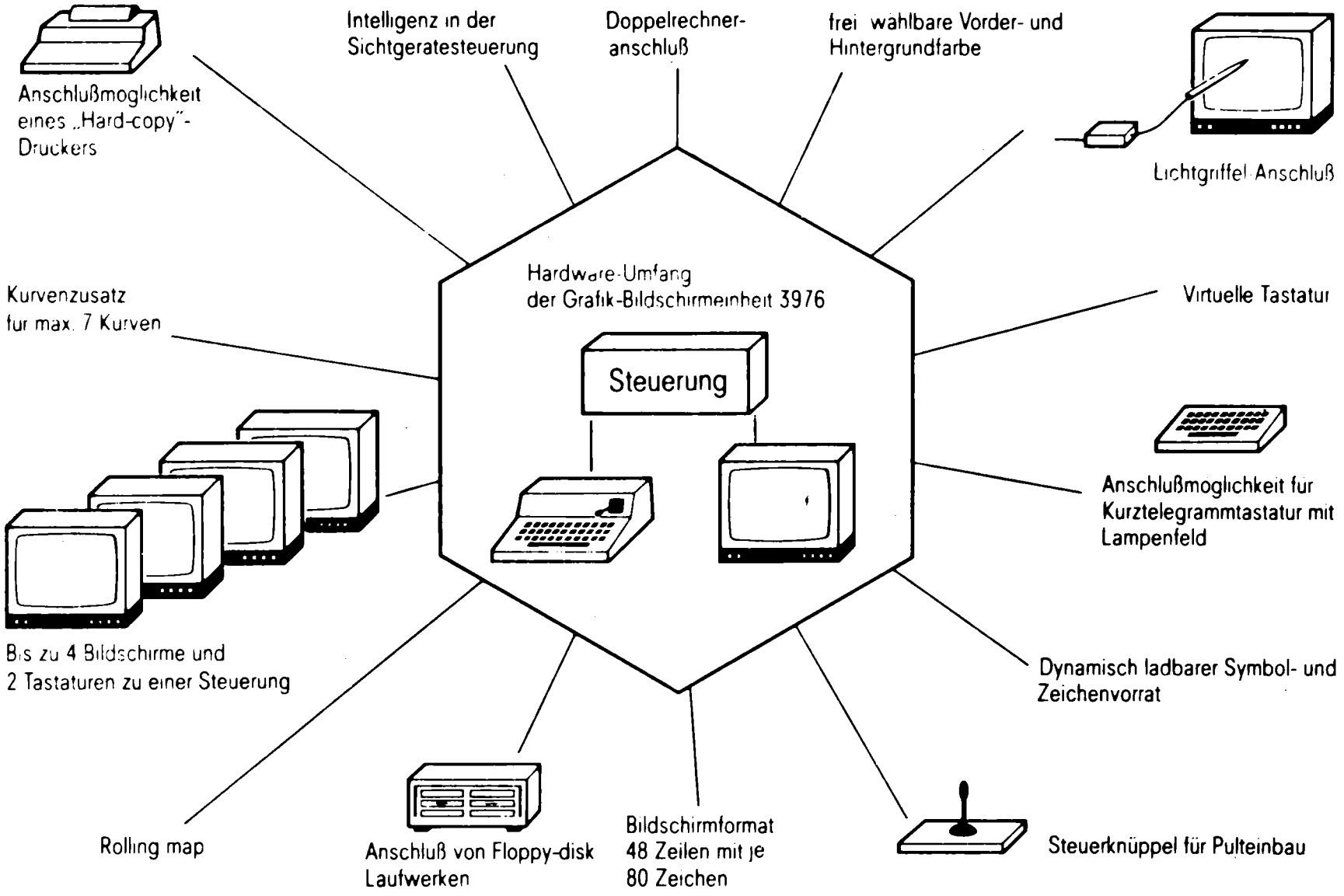
Zentralspeicherrahmen



Anpassung der verschiedenen Gerätefunktionen und -Schnittstellen an die Ein- Ausgabe -Anschlußstelle der Siemens Systeme 300 und einen universellen Gerätetreiber in der Grundsoftware mittels

Firmware

= programmierter Mikroprozessor



E STE4

Neues Prozeß-Sichtgerätesystem

Verbesserungen gegenüber der Grafik-Bildschirmeinheit 3976

Bild 11

PEARL 300 : Kompiliersystem und Verwendung für Echtzeitaufgaben

H. Schoknecht

- Einleitung

- Kompiliersystem PC 30
  - Entwicklungsziele
  - Arbeitsweise / Implementierung
  - Speicherplatz
  - Erstellungsweg für PEARL-Programme

- Sprachmittel für Echtzeitaufgaben

- Erfahrungen

- Portabilität von PEARL-Programmen

- Ausblick

Einleitung

Bereits 1977 wurden auf der GMR-GI-GfK Fachtagung Prozeßrechner in Augsburg /1/ und auf der 8. Jahrestagung des Siemens Anwender-Kreises SAK /2/ der Stand der PEARL-Implementierung auf PR330 vorgestellt.
Für die Sprache PEARL möchte ich auf die einschlägigen Veröffentlichungen /3,4/ hinweisen und Ihnen im folgenden kurz das Kompiliersystem PC30 für PEARL 300 /5,6/ vorstellen. Neben einem allgemeinen Überblick soll auch auf Implementierungsdetails wie der Lexikalischen Analyse, Namenselimination und Codeerzeugung eingegangen werden.

Nach Auflistung der vorhandenen Sprachmittel für Echtzeitaufgaben wird dieser Vortrag durch Darstellung uns bekannter Erfahrungen bei der Programmierung mit PEARL abgerundet werden, gefolgt von einigen realistischen Betrachtungen zum Thema "Portabilität von Programmen".

Am Ende wird ein kurzer Ausblick auf die Zukunft stehen.

Kompiliersystem PC30

Aufgrund der schon in 1976 bei Siemens mit einer Pilotimplementierung von PEARL gemachten Erfahrungen /7/ wurden folgende Entwicklungsziele für das Produkt PC30 festgelegt. Um Mißverständnissen vorzubeugen sollte betont werden, daß mit der lexikalischen Reihenfolge bei der Aufzählung keine Aussage über Prioritäten verbunden ist.

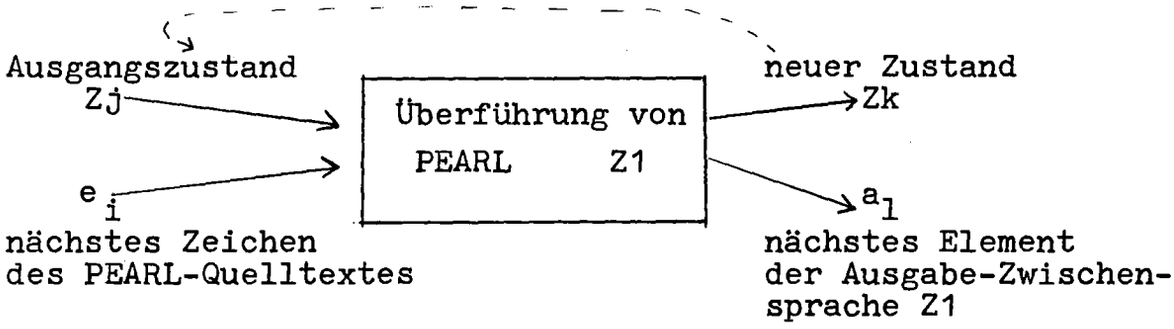
- Entwicklungsziele des PC 30

  - Ablauf unter ORG K
  - Einbettung in das leistungsfähige Siemens-System 300-16 Bit
  - Gute Fehlerdiagnose
  - Integrierte Testmöglichkeiten für Objektprogramme
  - Kurze Kompilierzeit
  - Laufbereich 16 K
  - Möglichst gute Codeerzeugung ohne zusätzlichen Optimierungslauf
  - Realisierung von Basis-PEARL

  Diese Aufgabenstellung und besonders die Forderung eines 16 K-Laufbereiches führte zu einem hochsegmentierten Programmsystem, bestehend aus 35 Einzelsegmenten, das hier in einer groben Aufgabengliederung (entsprechend den sieben Verarbeitungspässen) möglichst transparent dargestellt werden soll (siehe Bild 1).

  Wie schon in der Einleitung angedeutet, soll wegen der Kürze der Zeit nur auf drei entwicklungstechnische Details eingegangen werden, die den hohen Stand der verwendeten Implementierungstechnik dokumentieren.

- Lexikalische Analyse des PEARL-Quelltextes mittels eines deterministischen Automaten



$Z_j, Z_k \in Z$, $e_i \in E$, $a_e \in A$

$Z_k = F(Z_j, e_i)$ F ist durch eine Überführungsmatrix definiert

$Z = \{ Z_j \;/\; Z_j \mathrel{\hat{=}}$ Zustand des Automaten$\}$

z.B.: Identifier, Dezimalzahl, Exponentenziffer,...
(27 verschiedene Zustände sind möglich)

$E = \{ e_i \;/\; e_i \mathrel{\hat{=}}$ Klassen von Zeichen des PEARL-Quelltextes$\}$

(Die Klasseneinteilung erfolgt aufgrund syntaktischer und semantischer Bedeutung der einzelnen Zeichen.

Alle (ASCII-)Zeichen werden in 13 Klassen abgebildet)

$A = \{ a_e \;/\; a_e \mathrel{\hat{=}}$ Element der Zwischensprache Z1$\} \vee \{\}$

z.B.: Identifier, Fixed-Zahl, Float-Zahl,...,
kein Eintrag
(Es gibt etwa 160 verschiedene Elemente in Z1)

- Namenselimination unter Berücksichtigung des Transferverhaltens von externspeicherresidenten Listen, zu denen über software-paging zugegriffen wird.

  Eine wichtige Teilaufgabe eines Compilers ist das Ersetzen von vereinbarten Programmgrößen an der (den) Anwendungsstelle(n). Da eine Einschränkung bzgl. Programmgrößen und Programmlängen dem Anwender nicht zumutbar ist, sind die entsprechenden Listen externspeicherresident zu führen und einem Software-paging zu unterwerfen.

  Bei einer Implementierung sind mehrere Verfahren möglich. Für den PC 30 haben wir ein sehr leistungsfähiges gewählt /1/, das sich in eine Listenphase (LIPH) und eine Eliminationsphase (ELPH) unterteilt (siehe Bild 2).

  Zunächst wird mit der Listenphase begonnen und wenn der für die Liste im HSP reservierte Platz (Page) erschöpft ist, erfolgt vor dem Austransfer auf ESP eine Eliminationsphase. Wie aus dem folgenden Bild ersichtlich, erstreckt sich die Eliminationsphase auf den möglichen Gültigkeitsbereich der im HSP stehenden Programmgrößen.

  Danach erfolgt eine weitere Listenphase usw.

- Codeerzeugung unter Berücksichtigung der verfügbaren Hardware-Befehle

  Die Codeerzeugung erfolgt unter Berücksichtigung von zwei Registersätzen und eines Kellers (als Ersatz für weitere Registersätze) durch Abarbeitung des im folgenden Beispiel (Bild 3) dargestellten Baumes von rechts nach links.

  Diese etwas unübliche Bearbeitung führt dazu, daß Zwischenergebnisse, die in Ermangelung weiterer Registersätze zu kellern sind, stets als rechter Operand eines dyadischen Operators auftreten. Da die Hardware des PR 330 für linken Operand im Register, rechten Operand im Keller spezielle Befehle vorsieht, ist ein explizites Entkellern nicht mehr notwendig, wodurch automatisch eine Optimierung erreicht wird.

- Platzbedarf

  Ausgeliefert wird der Compiler auf Platte und belegt dort folgenden Speicherplatz.

Ausgelieferter PC30	160 KW
Ausgelieferte Laufzeitroutinen	80 KW
Vom Ladebinder erstellte Version (mit einfacher Listenlänge, Page=1)	180 KW
Laufbereich im HSP	16 KW

- Erstellungsweg für PEARL-Programme

  Die drei Aufgaben zur Erstellung eines PEARL-Programms Compilieren, Binden, Laden oder Lade-Binden (siehe Bild 4) sind Grundlage unserer Übersetzungstechnik für alle Compiler.

Sprachmittel für Echtzeitaufgaben und EIA-Verkehr

Für das Arbeiten mit TASK's (Programme im Sinne des Betriebssystems) stehen dem Anwender folgende Aufrufe zur Verfügung:

Funktion	Beispiele in PEARL
Starten	ACTIVATE T;
Starten mit Schedule	ALL 3 HRS UNTIL 12:00 ACTIVATE T1;
Beenden	TERMINATE T2;
Fortsetzen	CONTINUE T3;
Fortsetzen mit Schedule	AT 5:00 CONTINUE T3;
Anhalten	SUSPEND;
Anhalten und Fortsetzen bei AL1	WHEN AL1 RESUME;
Löschen aller Einplanungen (Schedules)	PREVENT;
Koordinierung	REQUEST SEM; RELEASE SEM;
Signals	ON SIG1: Signalreaktion;
Erzeugen eines Signals für Testzwecke	INDUCE SIG1;
α-num Eingabe	GET A,B,C FROM LKEI BY(3)F(7,1);
α-num Ausgabe	PUT A,B,C TO DRUA BY(3)F(5,2);
binäre Eingabe	TAKE BGR FROM DIGEIN ;
binäre Ausgabe	SEND BGR TO DIGAUS ;
Typgebundene Eingabe	READ A FROM DATEI ;
Typgebundene Ausgabe	WRITE A TO DATEI ;

Erfahrungen mit PEARL

Von Herrn Professor Dr. Lauber wurden 1977 in Augsburg /8/ folgende Zahlen vorgestellt, die einen Vergleich PEARL-Assembler beinhalten:

	Faktor
• Entwicklungszeit eines Programmes einschließlich Test	0,2 (= 20%)
• Laufzeitverlängerung	1,2 - 1,5

Hierbei handelt es sich um pauschalierte Aussagen, wobei die Einzelergebnisse durchaus weit streuen können (angegebener Extremfall bei Entwicklung 3%). Aber auch der Assembler-Programmierer liefert unseren Erfahrungen nach Ergebnisse, die eine Streuung bis zum Faktor 7 zulassen.

Die Stärken einer höheren Programmiersprache liegen für den Anwender generell in der Verkürzung der Entwicklungszeit, einer prägnanten und kompakten Darstellung und damit in einer Erleichterung der Programm-Dokumentation und -Wartung. Mit PEARL sind nun auch Echtzeit-Vorgänge in einer höheren Programmiersprache formulierbar.

Jedoch muß hierfür - und das wollen wir ganz klar sagen - mit einer Code- und Laufzeitverlängerung gerechnet werden.

- Assembler-Anschluß

  Mancher Anwender findet es als zu starke Einschränkung, wenn er auf den (in PEARL bewußt nicht realisierten) Assembleranschluß verzichten muß. Die von uns geschaffene Nahtstelle über Prozedurspezifikation und -Aufruf hat sich als äußerst hilfreich erwiesen.

Portabilität von PEARL-Programmen

Das Ziel der Sprachentwickler war das Erreichen einer möglichst hohen Portabilität eines PEARL-Programmes, was zu einer Trennung der systembezogenen und problembezogenen Daten führte.
Ein Anlagen- oder Gerätewechsel erfordert somit nur eine Änderung der systembezogenen Daten (Systemteil).

Ein solches Vorgehen wird u.E. in dieser einfachen Form nur innerhalb der gleichen Rechnerfamilie möglich sein, denn besonders Echtzeit-Programme können bzgl. ihrer Laufzeit sehr empfindlich reagieren beim Ablauf auf einem anderen Rechner mit geänderter Bitstruktur, Zykluszeit oder einer unterschiedlichen Peripherie. Auch ist es hierbei wichtig, welcher Subset von PEARL auf dem Ziel-Rechner realisiert ist.

Aus heutiger Sicht wird ein PEARL-Programm ohne weitere Änderungen nur dann auf einen anderen Rechner übertragbar sein, wenn der dort realisierte Sprachumfang maximal den Durchschnitt aller vorhandenen Implementierungen umfaßt. Dies wird Basis-PEARL sein.

Die Forderung nach einer strengen Portabilität für ein Echtzeit-Programm ist somit nicht ohne Rückwirkung auf die Hardware und auch auf den Problemteil zu sehen.

Ausblick

Das 9/77 festgeschriebene Basis-PEARL /3/ soll bis Mitte 78 genormt sein und einen verbindlichen Standard von PEARL bezeichnen. An eine Normung von full PEARL /4/ ist gedacht.

Heute ist der PC 30 als Produkt auf PR 330 verfügbar. Zur Zeit wird dieser PC 30 auf Prozeßrechner R 40 realisiert und Ende 78 verfügbar sein.

PEARL kann sich damit bei unseren vielfältigen Aufgaben auch auf internationaler Ebene bewähren und helfen Erfahrungen zu sammeln. Jedoch ist zusätzlich die Rückkoppelung vom Anwender notwendig, inwieweit Basic-PEARL für Echtzeitaufgaben als geeignet und ausreichend angesehen wird.

Auf der Basis dieser Erfahrungen wird sich entscheiden, ob und in welchem Maße Erweiterungen in Richtung full PEARL notwendig sind und evtl. auftretende Probleme besser lösen.

/1/ Rieder, P.: Effiziente PEARL-Implementierung für den PR 330. GMR-GI-GfK Fachtagung Prozeßrechner 1977 (Augsburg), Springer-Verlag.

/2/ Degelow, L.; Gottwald, H.-J.; Schoknecht, H.: Stand der PEARL-Entwicklung. 8.Jahrestagung des Siemens-Anwenderkreises SAK 1977, Fachhochschule Dortmund, S. 73-89.

/3/ Basic PEARL Language Description. Druckschrift der Gesellschaft für Kernforschung mbH, Karlsruhe 1977. Forschungsbericht KFK-PDV 120.

/4/ Full PEARL Language Description. Druckschrift der Gesellschaft für Kernforschung mbH, Karlsruhe 1977. Forschungsbericht KFK-PDV 130.

/5/ PEARL 300. Programmbeschreibung. Siemens AG, Bestell-Nr. P71100-D3010-X-X-35.

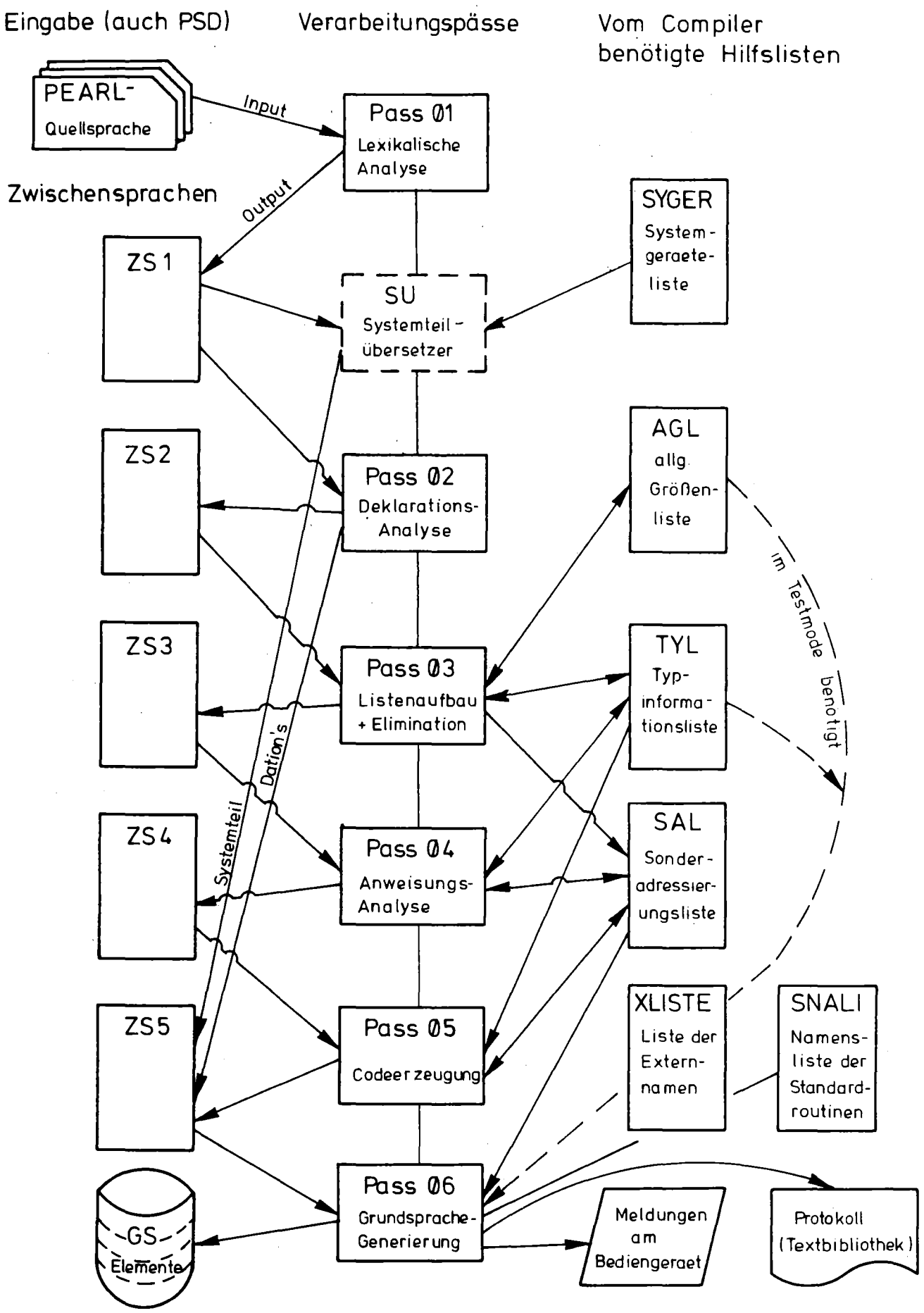
/6/ Dorn M.; Wenzel, T.: Prozeßsprache PEARL 300 für die Siemens-Systeme 300. Siemens-Zeitschrift 52 (1978) Heft 1, S. 23-27.

/7/ Gottwald, H.-J.; Schoknecht, H.: PEARL-Eine leistungsfähige Echtzeit-Programmiersprache. Regelungstechnik (1978) Heft 1, S. 23-27.

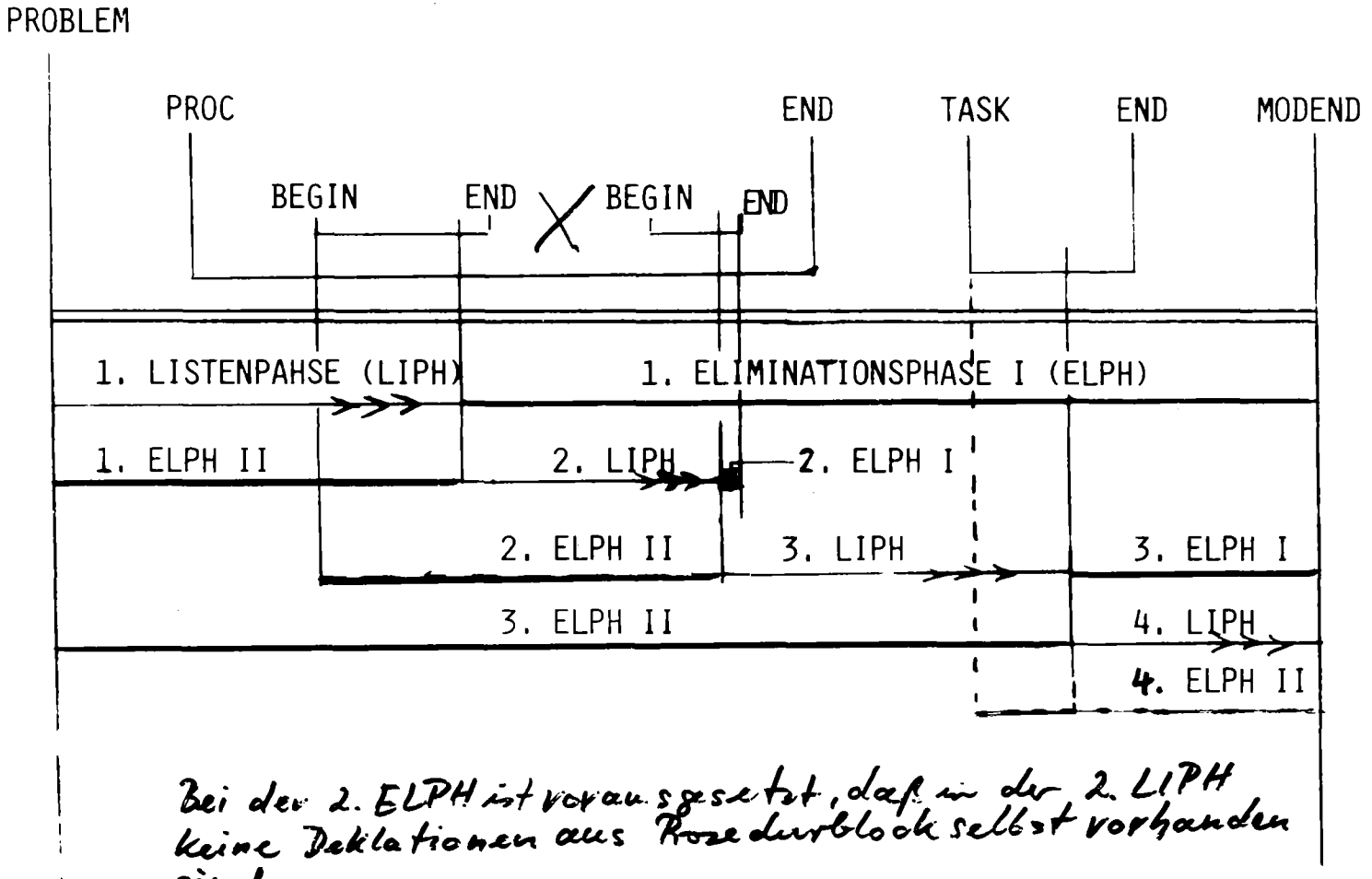
/8/ Lauber, R.: Experimentelle Untersuchung von Spracheigenschaften der Prozeßsprache PEARL anhand von Modellprozessen, aus "Praxis von Sprachen, Programmiersystemen und Programmgeneratoren", Hauser Verlag 1976, S. 13-30

Bild 1 / E STE 33

Struktur des Compilers PC 30



BEISPIEL ZUSAMMENSPIEL LISTENPHASE-ELIMINATIONSPHASE



Bei der 2. ELPH ist vorausgesetzt, daß in der 2. LIPH keine Deklarationen aus Rosedurblock selbst vorhanden sind.

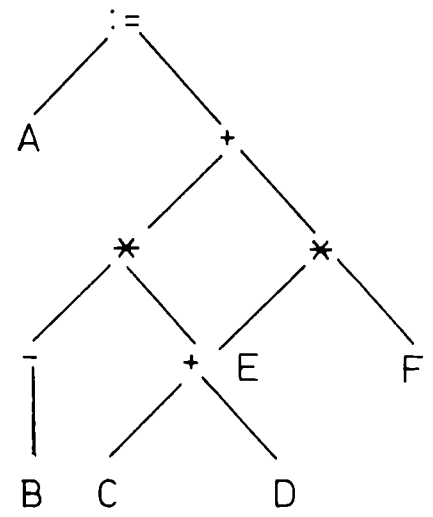
Beispiel einer Codeerzeugung

(1) A: = - B * (C + D) + (E * F); (A,B,C,D,E,F seien Fixed-Zahlen)

Darstellung von (1) als Postfix-Form

A,B,-,C,D, +, *,E,F, *,+, : =

Baumstruktur

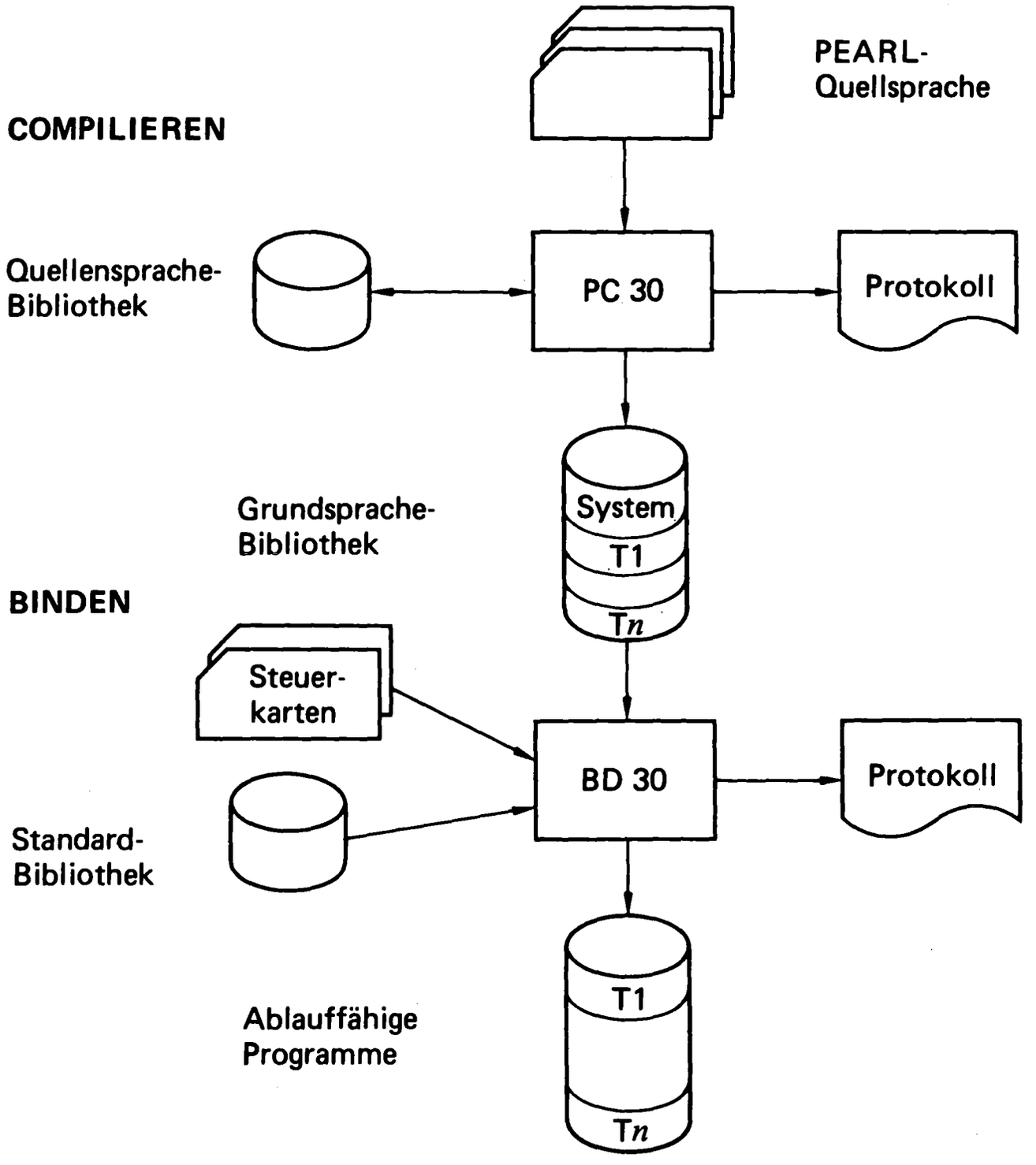


Abarbeitungsrichtung für Codeerzeugung unter Berücksichtigung der Operatorprioritäten.

Codeerzeugung (in Assemblernotation):

'A'	G1 : = (E)	
'F'	G1 : = G1 * (F)	*** E * F
'A'	G5 : = (C)	
'F'	G5 : = G5 + (D)	*** C + D
'A'	('R2) : = G1	
'A'	G1 : = (B)	
'F'	G1 : K G1	*** - B
'F'	G1 : = G1 - G5	*** - B * (C + D)
'F'	G1 : = G1 + (R2')	*** - B * (C + D) + (E * F)
'A'	(A) : = G1	*** A : = . . .

Bild 4 Erstellungsweg für PEARL-Programme

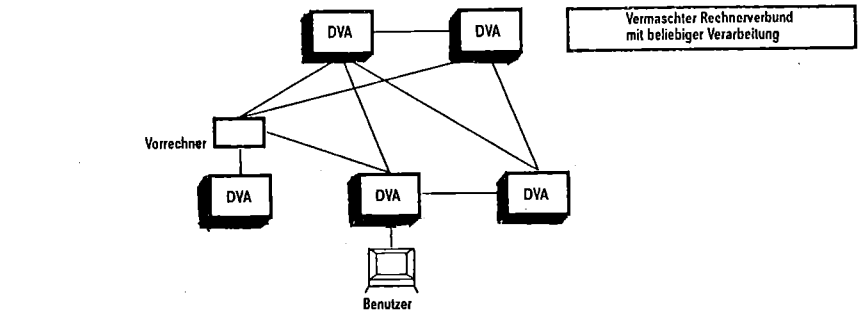


K.-H. Drawert, K. Pieper

Daten- und Funktionsverbund mit Rechner des Siemens-Systems 300 - 16 bit

---

Außer den beiden als Thema genannten Verbundarten können noch zwei weitere genannt werden:



Datenverbund

Benutzer einer Anlage greift auf Daten zu, die in anderer Anlage gespeichert sind.

Wichtig für Auskunftssysteme, Platzbuchungssysteme, Dokumentationssysteme.

Funktionsverbund

Benutzer einer Anlage nutzt Hard- und Softwarefunktionen einer anderen Anlage z.B. Plotter, PASCAL-Compiler.

Sehr aktuell im Hochschul- und Großforschungsbereich.

Lastverbund

Wenn eigene Anlage überlastet, wird Auftrag in anderer Anlage bearbeitet. Benutzer merkt davon nichts.

Von Interesse z.B. für DVA-Dienstleistungsunternehmen.

Sicherheitsverbund

Ausfall einer Anlage wird von anderen Anlagen aufgefangen.

Schwerpunktanwendungen eines vermaschten Rechnerverbundsystems

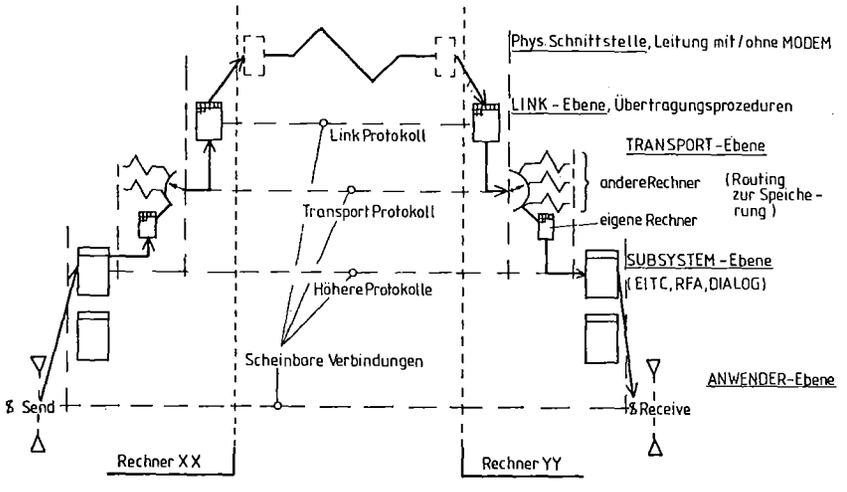
Von einem Sicherheitsverbund spricht man, wenn bei Ausfall eines Rechners seine Aufgaben von anderen durchgeführt werden.

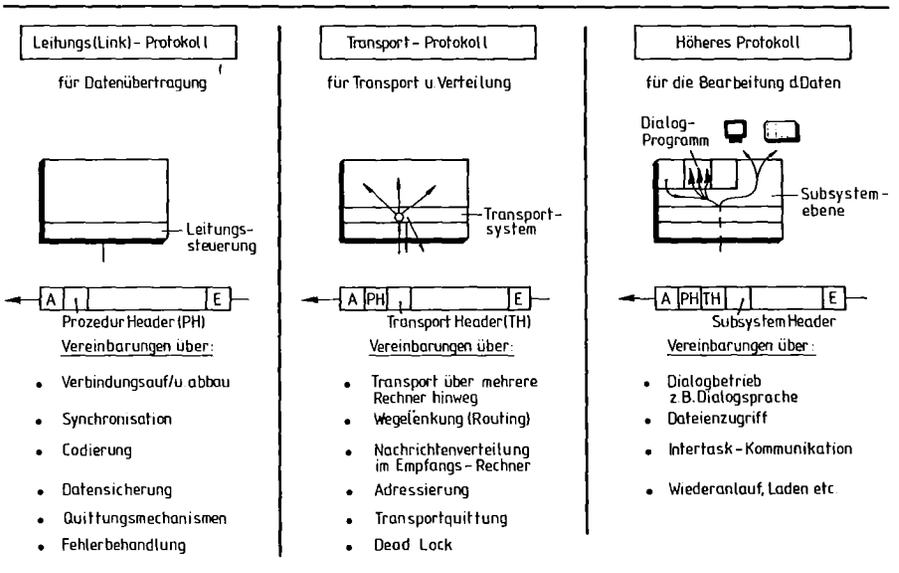
Bei einem Lastverbund werden von anderen Anlagen die Aufgaben übernommen, die auf einer überlasteten Anlage nicht ausgeführt werden können.

Der Funktionsverbund ermöglicht den Zugriff zu Hardware- und Softwarefunktionen ferner Rechner. Funktionen wie DIALOG, RDA (remote data access), RPC (remote program control) werden dieser Verbundart zugerechnet.

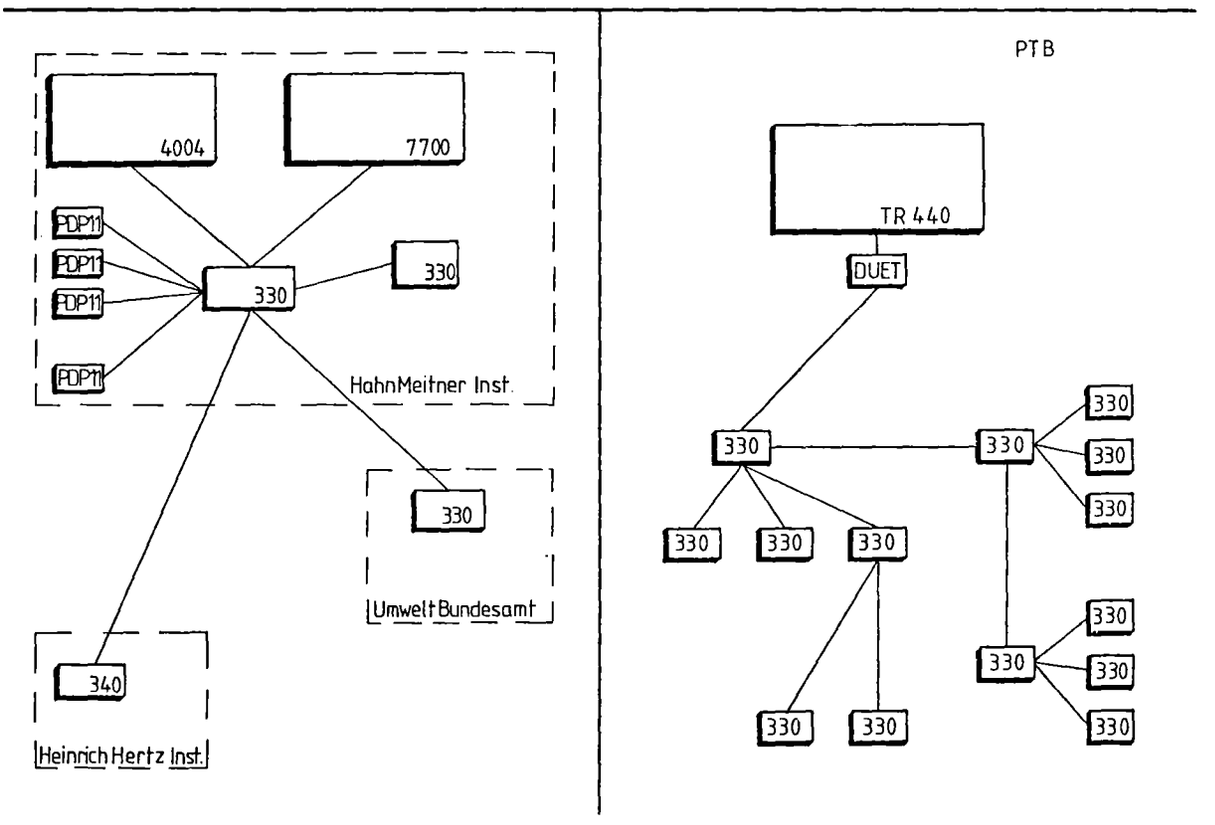
Der Datenverbund ist Grundlage jeder anderen Verbundart. Er wird durch die Funktionen des Subsystems EITC (extended intertask-communication) repräsentiert.

Innerhalb eines Rechnerverbunds gibt es mehrere Datenübertragungsebenen. Jede Ebene ist definiert durch ihre Stellung im Datenaustauschprozeß.





Bei Kommunikationen zwischen Komponenten verschiedener Datenverarbeitungsprozesse muß auch unter Randbedingungen ein sicherer Informationsaustausch gewährleistet sein. Randbedingungen sind in einem heterogenen Netz (z.B. Hahn-Meitner-Institut) vor allem die unterschiedlichen Konfigurationen der Partner, ihre sehr unterschiedlichen Betriebssysteme und die stark voneinander abweichenden Einzelaufgaben.



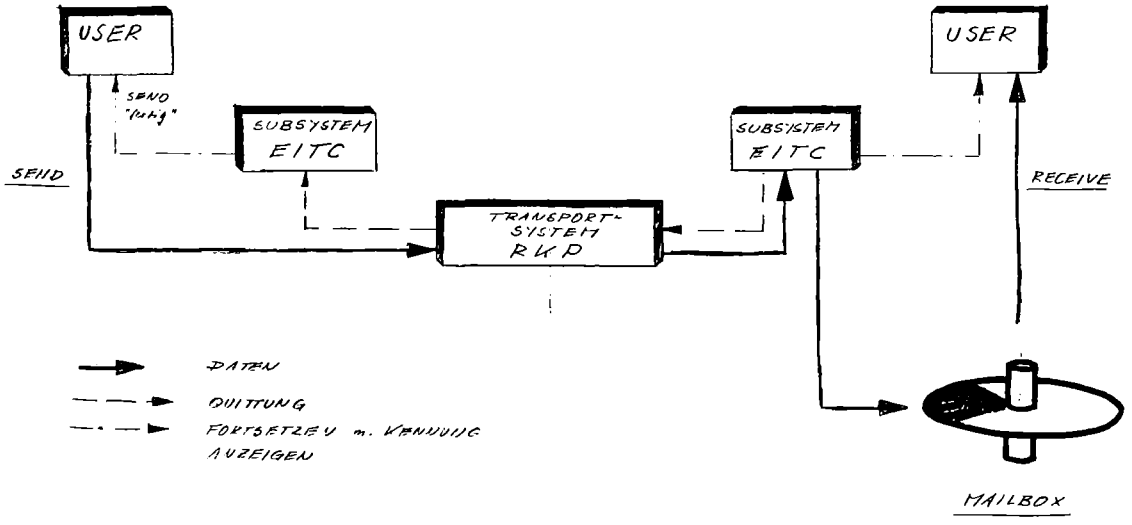
Datenverbund

Das Subsystem EITC dient zum Datenaustausch zwischen Anwenderprogrammen.

Das Senden und Empfangen von Daten kann zu jedem beliebigen Zeitpunkt ohne Rücksicht auf die Netzauslastung erfolgen.

Beim Senden einer Nachricht gelangen die Daten aus dem anwenderspezifischen Sendepuffer über das Rechnernetz in die sogenannte Mailbox des Empfängers.

Aus dieser Mailbox kann der Empfänger die Daten in seinen Datenpuffer lesen.

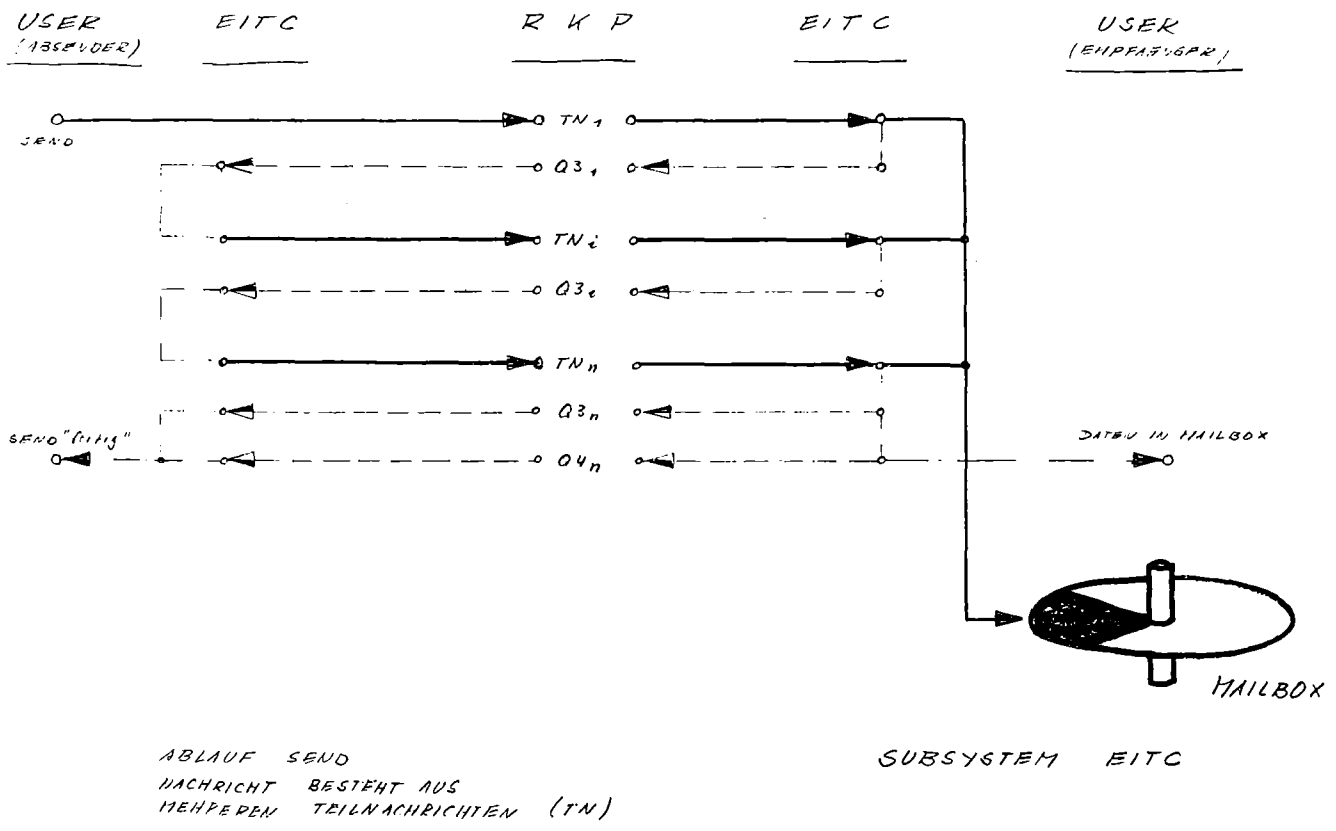


SUBSYSTEM EITC

STRUKTUR

Jeder Anwender hat eine Mailbox. Sie wird durch die anwendereigene Mailbox-Buchführung und durch Datenbereiche auf dem Externspeicher repräsentiert. Jede Nachricht belegt ein Buchführungselement und wird in einer Datei abgelegt. Da die Datei "beliebig" lang sein kann, wird die Größe der Mailbox in erster Näherung durch die zur Verfügung stehenden, hauptspeicherresidenten Buchführungselemente bestimmt.

Da es häufig vorkommt, daß die zu übertragende Datenmenge größer ist als die Datenmenge, die mit einem Aufruf der Transportebene übertragen werden kann, muß vom Subsystem EITC die Nachricht beim Senden in Teilnachrichten unterteilt werden. Jede Teilnachricht wird einzeln an den Empfänger gesendet. In der Mailbox werden die Teilnachrichten wieder zu der kompletten Nachricht zusammengesetzt. Zu Prüfungszwecken werden die Teilnachrichten numeriert. Dem absendenden EITC wird jede einzelne Teilnachricht quittiert. Ist vom sendenden Anwender eine Quittung über das ordnungsgemäße Ablegen der Nachricht in der Mailbox gewünscht, wird diese Quittung nach dem Eintreffen der letzten Teilnachricht an den Absender geschickt.



Das Anwenderprogramm, das Daten erwartet und aus der Mailbox zu lesen wünscht, gibt einen entsprechenden Aufruf. Die Daten werden aus der Nutzdatendatei der Mailbox in den vom Anwender angegebenen Datenbereich übertragen.

Die Empfangsfunktion kann eine komplette Nachricht in kleinen angebbaren Stücken aus der Mailbox lesen.

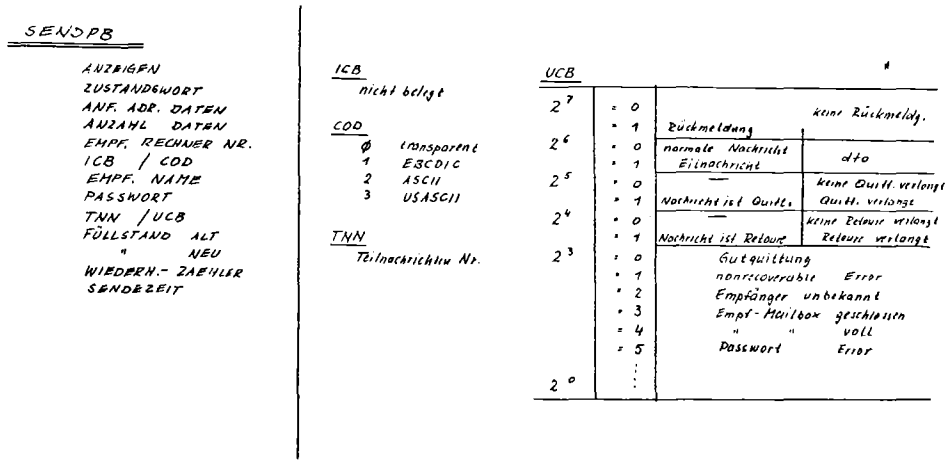
Nach jedem Lesen wird der Parameterblock des Aufrufs mit aktuellen Parametern versorgt. Dies sind z.B. die Absender-Rechnernummer, Absendername und die Anzahl der noch verbleibenden Datenbytes.

Der Empfangsaufruf kann auch die Daten eines angegebenen Absenders gezielt aus der Mailbox lesen.

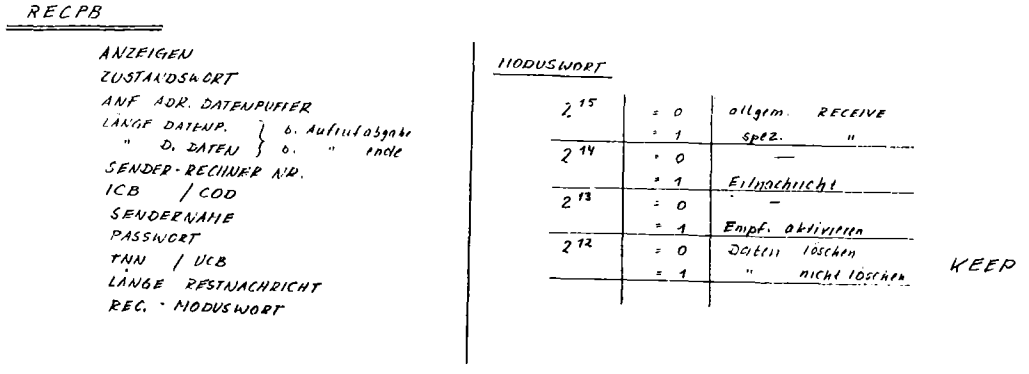
Sollten zum Zeitpunkt des Aufrufs die gewünschten Daten noch nicht in der Mailbox vorhanden sein, so wird der Wunsch vermerkt. Treffen dann die Daten ein, wird der Anwender davon benachrichtigt. Er kann dann die Daten lesen.

Jedes Programm, das am Netzverbund teilnehmen will, muß sich dem Transportsystem bekanntmachen. Dies geschieht durch einen entsprechenden Aufruf. Ein entsprechender Aufruf beendet die Kommunikation und das Programm. Dazu müssen alle Sende- und Empfangsaufrufe abgeschlossen sein.

Die Sende- bzw. Empfangsfunktionen werden durch entsprechende Aufrufe ausgelöst. Die Aufrufe werden mit der Adresse eines Parameterblocks versorgt. Aus den folgenden Bildern sind die Parameter der entsprechenden Aufrufe zu erkennen.



Parameter der Sendefunktion



Parameter der Empfangsfunktion

Zu den Parametern, die der Anschlußfunktion angegeben werden, gehört die Stations-Nummer. Jedes Anwenderprogramm und jedes Gerät ist im Rechner durch eine Stations-Nummer bekannt. Ein Anwender schließt sich also mit einer solchen Nummer an. Es wird geprüft, ob die Nummer schon für ein anderes Programm oder ob die Nummer beim Erstellen des Systems für ein Gerät in diesem Rechner vergeben wurde. Die Stations-Nummer ist Bestandteil des eigenen Programmnamens. Mit diesem Namen kann das Programm adressiert werden. Weitere Parameter der Anschlußfunktion sind die maximale Länge der zu empfangenden Daten, die maximale Anzahl gleichzeitig tätiger Netzaufrufe und ein Passwort. Dies dient zur Prüfung der eintreffenden Daten. Jeder Absender muß das Passwort des Empfängers angeben. Besteht keine Übereinstimmung, wird der Empfang der Nachricht abgelehnt und dem Absender eine entsprechende Anzeige in seinen Parameterblock eingetragen.

Funktionsverbund

Das Subsystem DIALOG ermöglicht das Aktivieren von Dialogfunktionen ferner Rechner in einem Rechnernetz.

Der Dialog kann von allen dialogfähigen Stationen erfolgen. Die Antwort wird an die Station ausgegeben, von der der Dialog angestoßen wurde.

Eine Station kann ein Gerät oder ein Programm sein.

Die Anzahl der gleichzeitigen Dialoge ist begrenzt. Es können also mehr Stationen angeschlossen sein als Sitzungen gleichzeitig möglich sind.

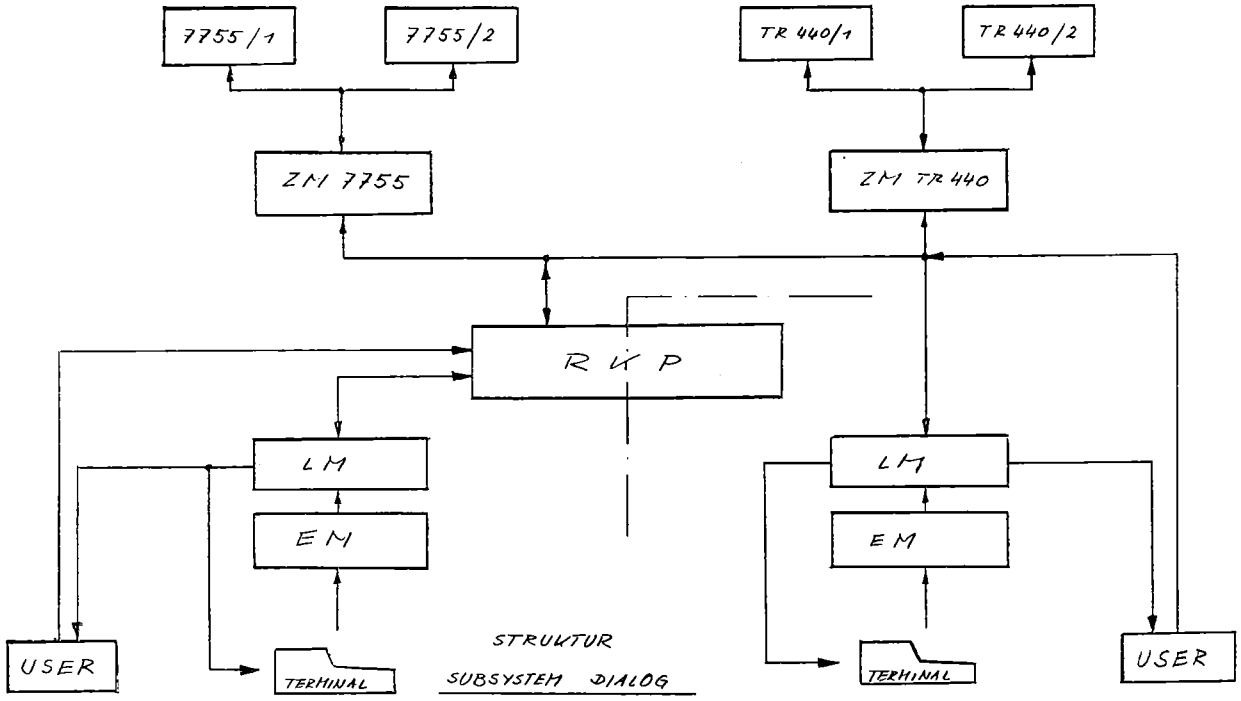
Um einen Dialog zu führen, muß von der Station eine Dialogsitzung eröffnet werden.

Während der Sitzung ist es möglich, Bedienungen an andere Ziele zu geben. Die können z.B. lokale bedienbare Programme, die Quittungsbearbeitung des Betriebssystems, das Standardbedienprogramm oder Dialogfunktionen anderer ferner Rechner im Netz sein.

Die Angabe einer Zielidentifikation gilt so lange, bis eine andere angegeben wird.

Das Subsystem besteht aus drei Funktionseinheiten:

1. Eingabemodul EM
2. Lokalmodul LM
3. Zentralmodul ZM

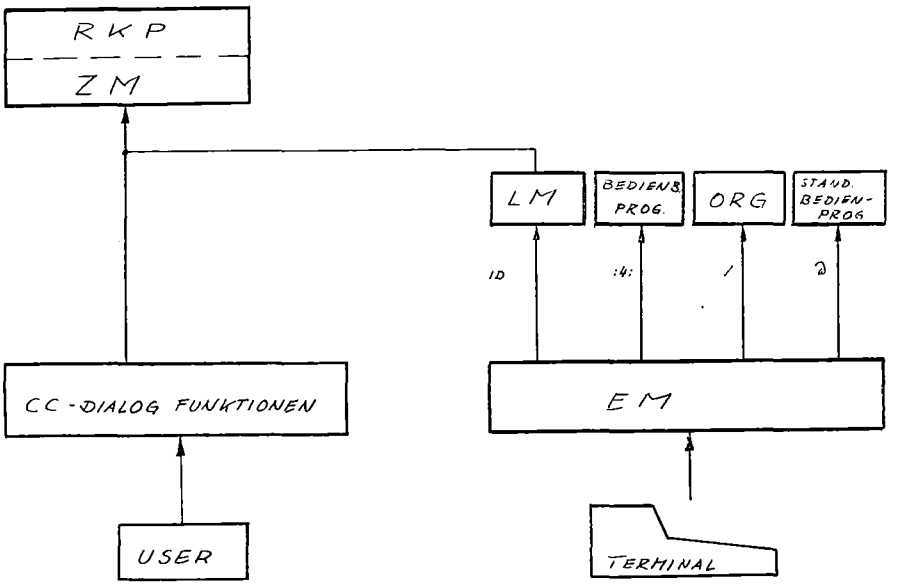


Der Eingabemodul untersucht die Eingabezeichenfolge auf Fluchtsymbole und übergibt die Zeichen an das gewählte Ziel. Dies kann das Standardbedienprogramm, die Quittungsbearbeitung des Betriebssystems, ein lokales bedienbares Programm oder, wenn als Ziel der Dialog-Rechner gewählt wurde, der Lokalmodul des Dialog-Subsystems sein.

Der Lokalmodul sendet den erhaltenen Text an den Zentralmodul, der dem gewählten Dialog-Rechner zugeordnet ist. Der Lokalmodul nimmt auch Quittungen und Texte vom Zentralmodul entgegen und gibt sie an die zuständige Station aus.

Der dialogrechnerspezifische Zentralmodul hat den Koppelverkehr abzuwickeln. Er muß die Texte auf logische Funktionen untersuchen. Dies kann eine Dialogeröffnung, das Beenden einer Sitzung oder der Abbruch des Dialogverkehrs sein. Dabei ist es gleichgültig, ob der Text von der Station oder vom Dialogrechner kommt.

Ferner muß der Zentralmodul die Texte auf Steuerzeichen untersuchen und sie gegebenenfalls konvertieren.

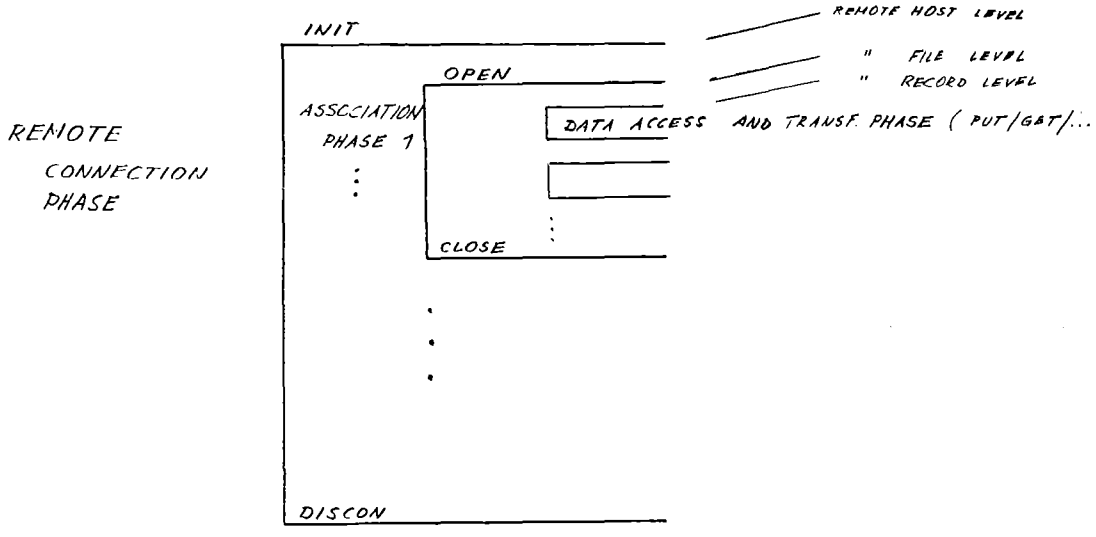


STRUKTUR DIALOG-SUBSYSTEM

Diese Aufteilung des Dialog-Subsystems wurde gewählt, um nicht in jedem Endrechner die aufwendigen Verwaltungsaufgaben des Zentralmoduls installieren zu müssen.

Für mehrere Dialog-Rechner gleichen Typs braucht der zugehörige Zentralmodul nur einmal im Netz installiert zu sein.

Bei Erweiterung des Netzes um einen oder mehrere Dialogrechner eines anderen Typs (Siemens 7755) braucht nur ein entsprechender Zentralmodul erstellt zu werden.



ABLAUFPHASEN

RDA

Das Subsystem RDA ermöglicht den Zugriff zu Geräten und Dateien in fernen Rechnern. Das System muß in jedem Rechner installiert sein, dessen Geräte dem Verbund zur Verfügung stehen sollen.

Alle Zugriffe zu Geräten und Dateien sind nur während einer Verbindungsphase möglich. Die Verbindung wird durch entsprechende Aufrufe hergestellt bzw. gelöst.

Das Subsystem ist in drei Funktionseinheiten gegliedert:

1. Auftraggeber

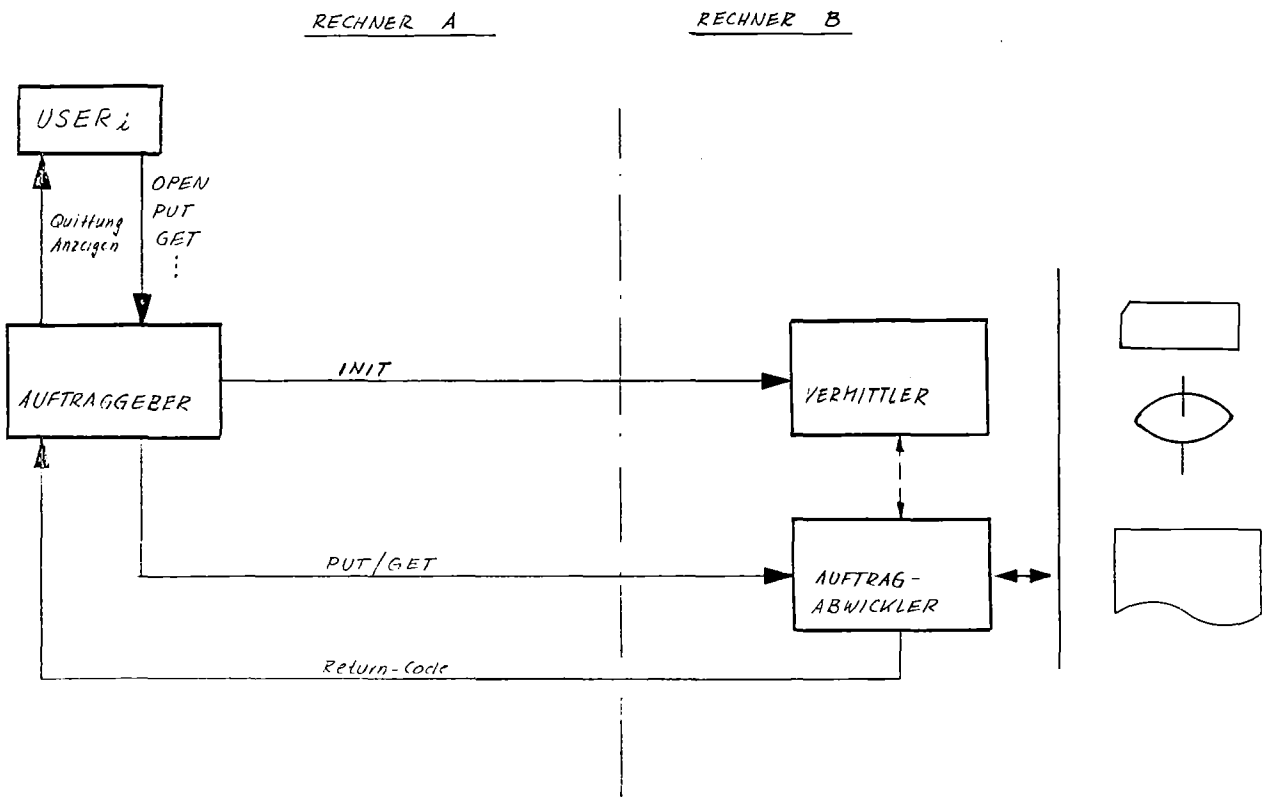
   Er gibt den Auftrag an den Partner im fernen Rechner weiter und nimmt Quittungen und Daten vom fernen Rechner entgegen.

2. Vermittler

   Er stellt die Verbindung vom Anwender zum Gerät bzw. zur Datei her.

3. Auftragsabwickler

   An ihn gehen sämtliche Aufträge vom fernen Rechner. Er gibt auf Geräte aus oder liest von Geräten und gibt die Daten an den Auftraggeber.



STRUKTUR
SUBSYSTEM RDA

STATISTISCHE ANGABEN ZUM STAND DER PROZESSDATENVERARBEITUNG IN DER BUNDESREPUBLIK DEUTSCHLAND

G.R. Koch
R.H. Hoffmann

Lehrstuhl für Planungs- und Programmiertechniken von Prozessrechnern (Prof. Dr. U. Rembold) am Institut für Informatik III der Universität Karlsruhe, Zirkel 2, 75 Karlsruhe 1

Inhalt: Ausgehend von der Frage, wie sich Prozessprogrammierung anwenderfreundlich gestalten ließe, werden Ergebnisse und Analysen einer Befragung von 65 Anwendern zu den Themen Rechnerauswahl, Projektierungserfahrungen, Erstellung oder Fremdkauf von Software und wirtschaftliche Rechtfertigung dargestellt.

Die Einsatzanteile von SIEMENS-, AEG und BBC-Rechnern sowie der Gruppe der "Sonstigen" werden für die bei den befragten Anwendern wichtigsten Klassen technischer Prozesse sowohl aktuell als auch im Trend angegeben.

Es bestätigt sich, daß bei den analysierten Projekten trotz erheblichen Planungsaufwandes die Kosten und die Fertigstellungstermine insbesondere für die Software im Schnitt zu über 50% überschritten wurden.

1. Einleitung

Trotz des starken Zuwachses an Prozessrechnerinstallationen (Bundesrepublik 1974 rd. 7000, 1976 rd. 12000; nach Diebold) findet man unter den Fachveröffentlichungen kaum übergreifende Berichte über Erfahrungen mit der Projektierung und den Einsatz von Prozessdatenverarbeitungsanlagen (PDVA), welche über die Darstellung einzelner spezieller Anwendungen hinausgehen /1/. Auch zusammenfassende Aussagen zum Stand der Entwicklung der PDV und zu Trends sind häufig sehr allgemein, nicht frei von spezifischen Marktinteressen und daher meist optimistischer als es sich in der Anwenderpraxis darstellt.

Die Kritik an der "Anwenderunfreundlichkeit" der Programmierung von Prozessrechnern und die häufig auftretenden Probleme bei der Projektierung von Automatisierungen mit PDV-Anlagen haben ihre Ursache meist in einer der beiden Grundhaltungen der Systementwerfer:

1. Die Entwerfer berücksichtigen benutzerorientierte Gesichtspunkte der Planung und Erstellung nur sehr unzureichend weil Sie mit ihrem System größtmögliche "technische Effizienz" zu erreichen versuchen. Dies erzwingt eine teure Anpassung des Anwenders an das System.

2. Die Entwerfer sind bemüht, sich in die Lage des Anwenders zu versetzen, modellieren aber in Wirklichkeit ungeprüft ihre eigenen subjektiven Vorstellungen von den Wünschen und Möglichkeiten des Anwenders. Dieses sehr verbreitete Verfahren bezeichnet J. Witt /2/ als das "Prinzip der vermeintlichen Wohltat".

Die vorliegende Arbeit versucht daher, die betroffenen Anwender von PDV-Systemen mit der Darstellung ihrer Erfahrungen beim Einsatz zentraler Prozessrechner zu Wort kommen zu lassen.

Die präsentierten Angaben wurden durch eine systematische Anwenderbefragung gewonnen. Dabei wurde mit der gleichen Methodik vorgegangen, wie sie eine amerikanische Fachzeitschrift für Prozeßsysteme schon 1971 anwandte.(Die kaum noch aktuellen Daten aus den USA werden in /3/ ohne Interpretationshilfen wie z.B.

Graphiken wiedergegeben.)
Einige Resultate von Umfragen in Teilbereichen (z.B. /4/) und Angaben von Herstellern wurden parallel ausgewertet. In keinem Fall ergaben sich Widersprüche zu eigenen Befunden.

Zu Beginn der Untersuchung war es nicht beabsichtigt, den Wirtschaftlichkeitsstudien der Anwender besondere Aufmerksamkeit zu widmen. Der Mangel einer konsequent angewandten Methodik der wirtschaftlichen Rechtfertigung war jedoch bald Anlaß, diesem Punkt breiten Platz einzuräumen.

2. Vorgehensweise bei der Datensammlung

Die Befragung des Anwenders erfolgte durch einen einheitlichen Fragebogen /5/, in ausgewählten Fällen auch in Form eines Interviews nach festem Muster. Der per Post verschickte Fragebogen sollte Auskunft geben über die eingesetzten Anlagen, ihren technischen Ausbau, verwendete Programmiersprachen und -hilfsmittel, Systemauswahlkriterien, Projektpersonal und Projektabwicklung. Von den ausgewählten 165 Anwendern haben 65 vollständig geantwortet, was einer Rücklaufquote von 40 % entspricht. Einige Antworten bezogen sich auf mehr als eine Installation, d.h. es wurden über 100 in der Bundesrepublik installierte Anlagen erfaßt, was knapp 1 % der technisch-wissenschaftlichen Rechner ausmacht, welche die Diebold-Statistik zu diesem Zeitpunkt verzeichnet. Vermutlich sind es weit mehr, da diese Statistik nicht unterscheidet, wie viele Anlagen tatsächlich zur Prozessautomatisierung eingesetzt werden.

Die Befragung dauerte 6 Monate und wurde in der zweiten Jahreshälfte 1976 abgeschlossen.

Zwanzig Anwender, von denen wegen des Umfangs ihrer Projekte oder der Vielfalt der verwendeten Systeme ein besonders qualifizierter Erfahrungsbericht erwartet werden durfte, wurden in einem je zweistündigen Gespräch anhand eines 90 Detailfragen umfassenden Katalogs befragt. Die damit oft hervorgerufenen Diskussionen vor Ort gaben Hinweise und Anregungen, wo die Schwerpunkte bei der Auswertung zu setzen seien.

3. Ergebnisse

Zur Unterteilung in Anwendungsgebiete für PDV-Anlagen wird die Prozessklassifizierung gemäß Bild 1 benutzt (vgl. auch /6/ und /7/). Nach dieser Einteilung ist in Bild 2 die Menge der erfaßten Anlagen nach Herstellern gegliedert. Die Bilder 3,4 und 5 beschreiben die Marktanteile der umsatzstärksten Prozessrechnerhersteller bei aktuellen und für konkret geplante Anwendungsvorhaben.

3.1. Hardware

Die Kernspeichergröße der erfaßten Prozessrechner hat einen Erwartungswert von knapp 600 K bit, das entspricht bei der am häufigsten verwandten Wortlänge von 16 bit etwa 36 K Kernspeichergröße. Die Standardabweichung liegt selbstverständlich sehr hoch.

Die Verfügbarkeit der Prozessrechner in den erfaßten Anwendungsfällen liegt bei einem Erwartungswert von 98,4 %. Dabei ist die Standardabweichung mit 1,68 auffallend niedrig.

3.2 Projektierung und Rechnerauswahl

Bild 6 zeigt die Präferenzen der Anwender bei der Auswahl von Prozessdatenverarbeitungsanlagen. Die Beteiligung der Prozessrechner-Anwender und der Prozessrechner-Lieferanten an der Durchführung der Projektphasen ist in Bild 7 dargestellt. Bild 8 demonstriert die unterschiedliche Dauer der Projektphasen, Bild 9 den benötigten Realisierungsaufwand. Hier fällt das zeitliche Übergewicht bei der Aufgabendefinition und der Programmierung ins Auge. Während sich die Dauer der Systemanalyse durch Aufstockung an Personal verkürzen und die Qualität der Ergebnisse verbessern ließe, ist das Problem bei der Softwareerstellung komplexer. Hier wird derzeit schon mit vergleichsweise hohem Einsatz an Personal gearbeitet.

Bild 10 und Bild 11 stellen veranschlagte und tatsächlich benötigte Realisierungszeit für Hardware und Software gegenüber.

Produktbeschaffenheit	homogen		diskret	
Prozesszeitverlauf	kontinuierlich	diskontinuierlich	kontinuierlich diskontinuierlich	diskontinuierlich
Prozessbezeichnung	kontinuierlicher Prozess	Chargenprozess	Stückprozess	Befehlsprozess
Prozessklasse	I	II	III	IV
Beispiele	Kraftwerktechnik Energieerzeugung Energieverteilung Hochofenprozess Eisenhüttenwerk Zementofenprozess Papierherstellung Grundstoffindust. Raffinerien Chemische Verfahrenstechnik	Oxygen-Stahlwerke Elektroöfen Verfahrenstechnik Analysenuntersuchungen Laborautomatisierung	Brammenverfolgung Stahlverarbeitung Lager- und Verteilsysteme in der Fertigungstechnik und im Handel Verkehrslenkung	An- und Abfahrvorgänge von Turbinen Ein- und Ausschalten von Pumpen und Gebläsen Prüffeldautomatisierung Nachrichtensammlung- und verteilung Raketenstart Raumfahrt

Bild 1: Prozessklassifizierung

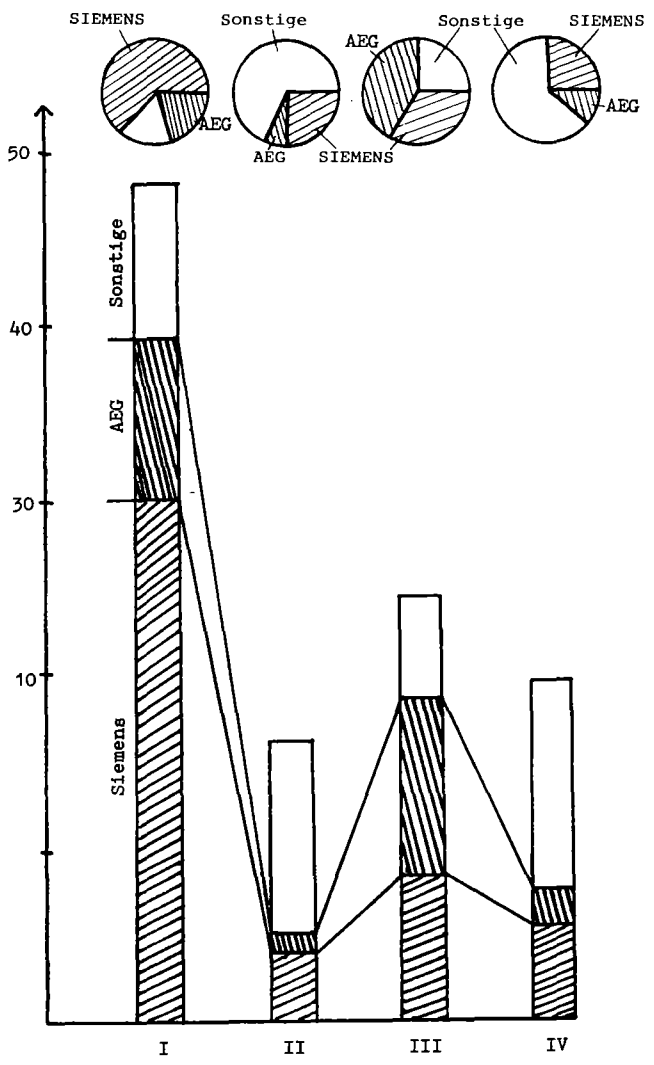


Bild 2: Erfaßte Prozessrechneranwendungen, geordnet nach den Anwendungsgebieten gemäß Bild 1 und nach Marktanteilen kumuliert

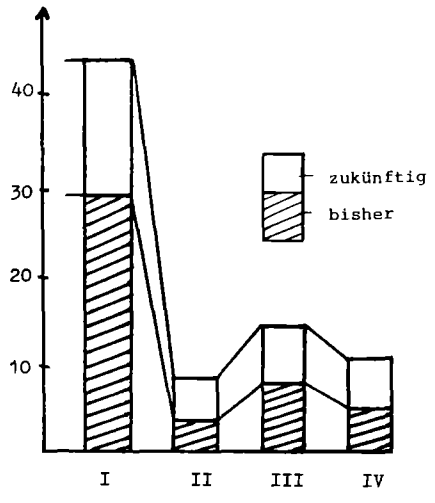


Bild 3: Bisheriger und für kommende Anwendungsfälle angegebener Einsatz von Siemens-Prozessrechnern

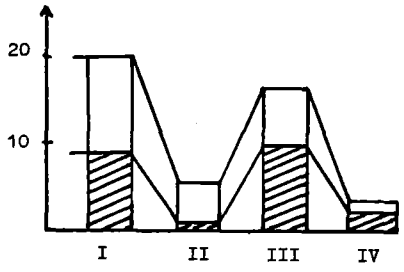


Bild 4: Bisheriger und für kommende Anwendungsfälle angegebener Einsatz von AEG-Prozessrechnern

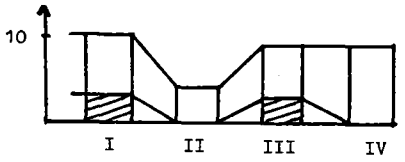


Bild 5: Bisheriger und für kommende Anwendungsfälle angegebener Einsatz von BBC-Prozessrechnern

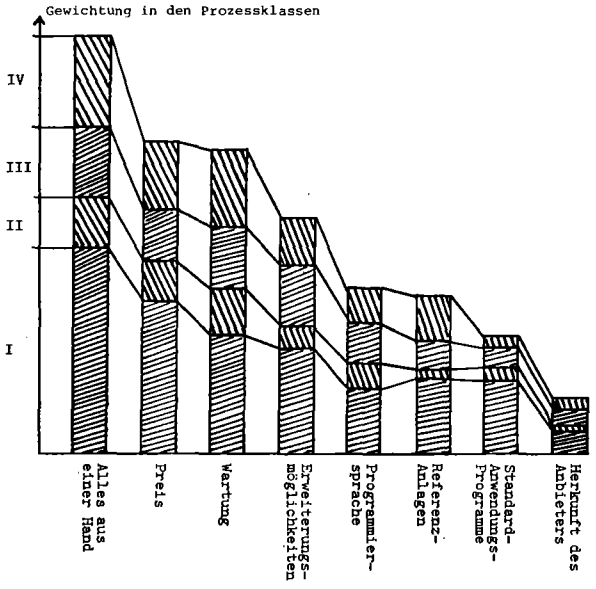


Bild 6: Gewichtung von System-Auswahlkriterien

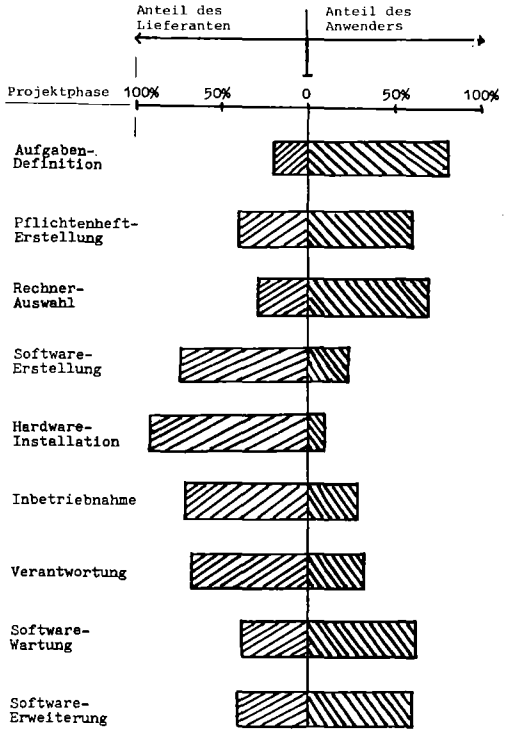


Bild 7: Beteiligung des Anwenders und des Lieferanten an der Durchführung der Projektphasen einer Prozessrechner-Installation

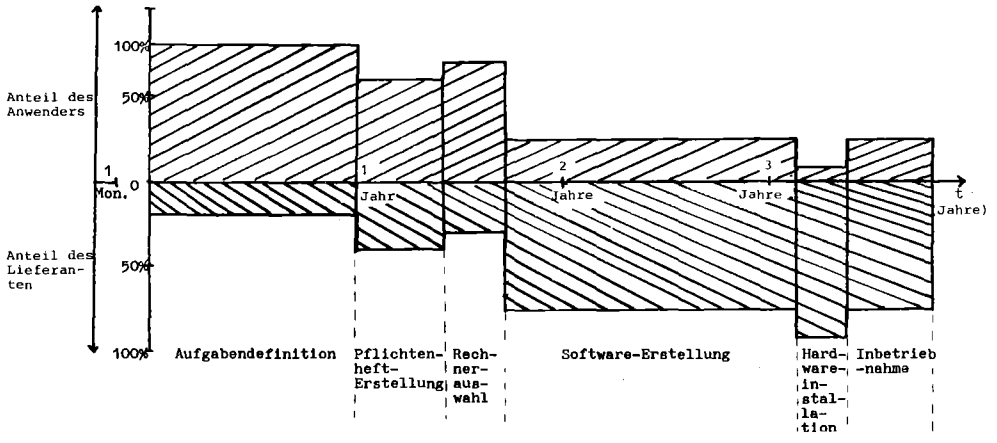


Bild 8: Dauer der Projektphasen

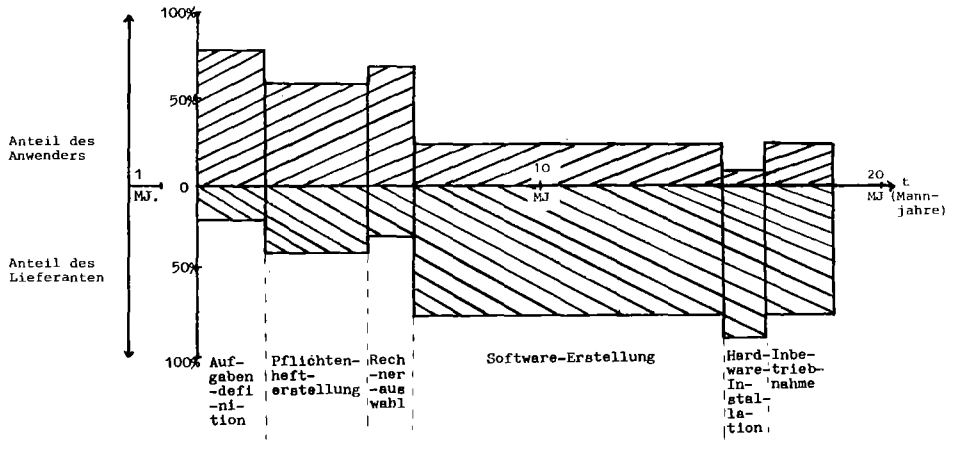


Bild 9: Aufwand für die Realisierung der Projektphasen

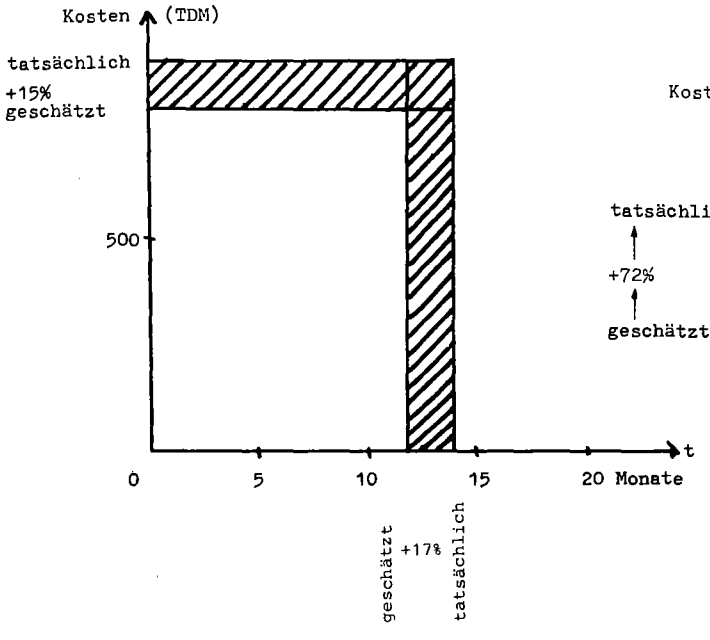


Bild 10: Hardware: Kosten und Realisierungszeit

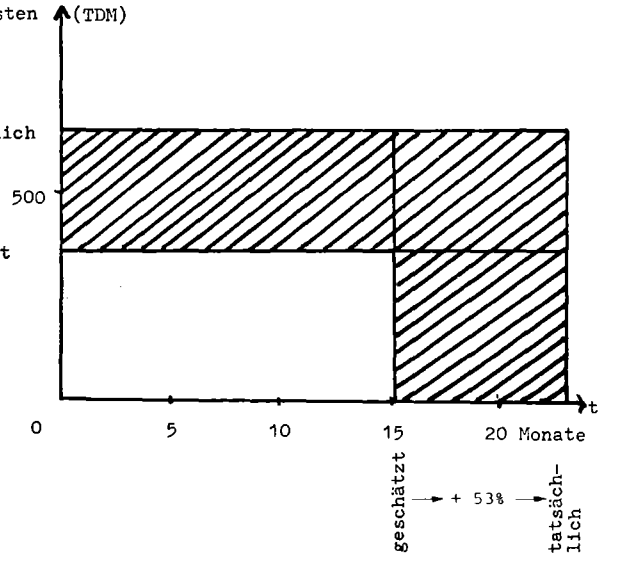


Bild 11: Software: Kosten und Realisierungszeit

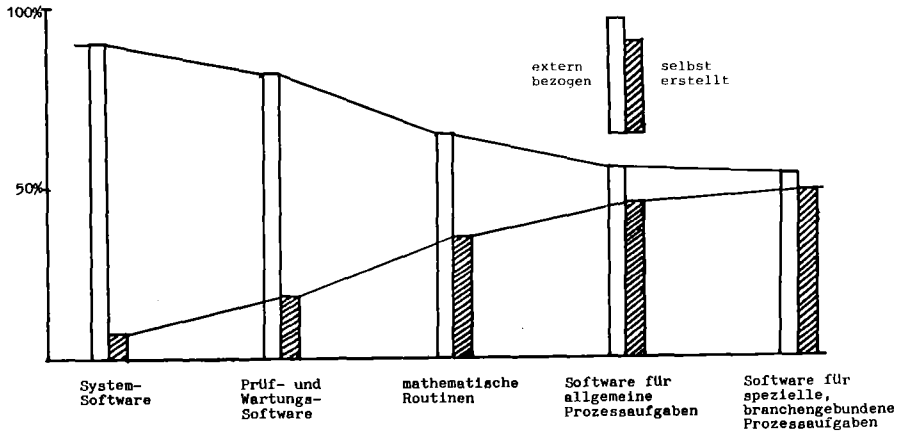


Bild 12: Vom Anwender gekaufte und selbsterstellte Software

3.3 Software

Aus einem Vergleich der von 22 Prozessrechner-Hersteller angebotenen Software /5/ geht hervor, daß jeder Lieferant seine Prozessrechner-Hardware mit einer Systemsoftware anbietet, die mindestens aus einem Organisationsprogramm und elementaren Dienstprogrammen besteht.

Auch läßt jener Vergleich erkennen, daß Programmierhilfen, Prüfprogramme für Hardware und Software sowie Programmbibliotheken für mathematische Verfahren meist mitgeliefert werden können. An vorgefertigten Programmen für allgemeine Prozessaufgaben wie Steuerung, Regelung , DDC, Meßwerterfassung und -verarbeitung und Betriebsdatenerfassung ist das Angebot geringer, ebenso verhält es sich mit dem auf noch weniger Prozessrechner-Anbieter verteilten Software-Angebot an Programmen für spezielle, branchengebundene Prozessaufgaben wie Laborautomatisierungen, Kraftwerkstechnik, Lagertechnik etc.

Mit dieser Feststellung korrespondiert auch das Befragungsergebnis bezüglich des Verhältnisses Software-Kauf zu Software-Selbsterstellung, dargestellt in Bild 12, welches allerdings noch nach Anwendungsgebieten zu differenzieren wäre. Wenn durch den Anwender Software gekauft wird, dann meist beim Prozessrechner-Lieferanten, der seinerseits häufig ein Software-Haus beauftragt. Die Mitgliedschaft in Anwenderklubs wird von den befragten Anwendern für wenig nützlich erachtet, da der Aufwand für die Anpassung übernommener Anwenderprogramme sich nur wenig kostengünstiger als die Neuerstellung eigener Programme ausnimmt, die zudem besser angepaßt sind.

Falls beim Anwender eine Beratung von außerhalb stattfindet, dann überwiegend unregelmäßig und bedarfsgesteuert durch den Lieferanten. Die Informationsfreudigkeit der Prozessrechner-Anbieter wird,von Ausnahmen abgesehen, für gut gehalten, die mitgelieferten Systemdokumentationen sind jedoch fast durchweg verbesserungsbedürftig. Hier ist ein wichtiger Ansatzpunkt zur Erhöhung der Anwenderfreundlichkeit von Prozessdatenverarbeitungssystemen zu finden. Tatsächlich bestehen solche Systemdokumentationen aus zusammengetragenen Fragmenten, die dazu noch in den

verschiedenen Hardware- und Software-Terminologien geschrieben sind und kein auf die Belange des Benutzers abgestimmtes Gebrauchsstück darstellen.

Höhere Programmiersprachen haben die bisher zur zeitkritischen Prozessprogrammierung verwendeten laufzeiteffizienten Assemblersprachen nur teilweise verdrängt. So werden 75 % der Anwendungsprogrammierung der erfaßten Prozessrechner-Einsätze mit Assemblersprachen bewerkstelligt, 25 % mit höheren Programmiersprachen. Das liegt zum einen daran, daß zum Zeitpunkt der Wahl des Prozessrechners eine Alternative nicht bestanden hat. Die Prozessrechner-Lieferanten boten fast ausschließlich Assembler, sowie seltener FORTRAN IV zur Anwendungsprogrammierung an. Zum anderen wird Assembler als die der Prozessautomatisierung adäquate Echtzeit-Programmiersprache verstanden. Dies ist vorwiegend bei einer Kategorie assemblererfahrener Programmierer der Fall, die höhere Echtzeit-Programmiersprachen ablehnen und auch die nächsten Jahre Assembler bevorzugen werden. Eine zweite Kategorie von Programmierern, bisher nicht mit Assembler tätig, würde gerne auf eine höhere Programmiersprache umsteigen, ist aber unsicher in Bezug auf den Zeitpunkt und die Art der höheren Prozessprogrammiersprache. Eine dritte Kategorie, die zahlenmäßig am kleinsten ist, verwendet von vornherein soweit als möglich höhere Programmiersprachen zur Prozessprogrammierung.

Dabei wird die Änderungsfreundlichkeit dort, wo höhere Programmiersprachen zur Anwendung kommen, einheitlich als "gut" bezeichnet, während das Urteil bei Assemblerprogrammen zwischen "sehr gut" und "gering", "aufwendig" und "umständlich" schwankt. Diese aufs erste uneinheitliche Beurteilung ist wieder durch die Unterscheidung in die beschriebenen Kategorien von Programmierern zu relativieren.

Auf die Frage, ob die Terminologie bei der Anwendungsprogrammierung deutsch oder englisch sein sollte, ist die Anwort eindeutig: Die Anwender würden das Deutsche bevorzugen, da dies eine Vereinfachung für die Benutzung der Anwendungsprogramme bedeute und eine immer wieder konstatierte psychologische Schranke beseitige. Nur im Falle der Vereinheitlichung durch eine universelle anwendbare Prozessprogrammiersprache wird Eng-

lisch akzeptiert, gleichfalls bei der Systemprogrammierung, die gemäß der Herkunft des Herstellers in einer nicht deutschen Sprache formuliert sein darf.

Die Fehlermeldungen werden in der Qualität unterschiedlich beurteilt. Erwünscht sind deutsche Klartext-Fehlermeldungen und die konsequente Trennung und die sofortige Klassifizierbarkeit von Systemmeldungen und Prozessmeldungen, um den Meldungsballast zu verringern. Gegenwärtig sind codierte Fehlermeldungen noch ebenso häufig zu finden wie Klartext-Fehlermeldungen.

Der hierarchische Aufbau der Anwendungsprogramme und die modulare Aufteilung nach dem Bausteinprinzip sind üblich und werden akzeptiert.

Durch Vorlage von sechzig Programmbausteinen bei 10 persönlich Befragten sollte ermittelt werden, welche Komponenten der Anwender für ein Programmiersystem für notwendig erachtet. Es handelte sich im wesentliche um Datenerfassungsprogramme, Datenaufbereitungsprogramme, Meßwertanalyseprogramme, Verarbeitungsprogramme, Datenausgabe- und Darstellungsprogramme, Systemprogramme, Dienstprogramme, Test-, Prüf- und Wartungsprogramme, Übersetzer. Dabei hat sich ergeben, daß

- die Prozessdatenverarbeitung durch das Programmgerüst Zeitgeber- und Zyklussteuerung, Zählwerterfassung, Meldungserfassung, Plausibilitätskontrolle, Grenzwertüberwachung, Mittelwertberechnung, digitale Filterung, Istwertberechnung, Zuwachsprüfung, Steuerung und logische Verknüpfungen sowie Sollwertvorgabe und Sollwertführung so gut wie vollständig abgedeckt wird,

- generell sehr großer Wert auf komfortable Bedienungs- und Korrespondenzprogramme einschließlich des Meldungs-, Protokoll und Dokumentationswesens gelegt wird,

- unerwartet großer Raum in den Systemprogrammen die gewünschten Hardware- und Software-Wartungs- und Testprogramme einnehmen,

- Assembler immer zur Verfügung stehen soll,

- PEARL allgemein bekannt ist, und als Prozessprogrammiersprache in der Mehrzahl der Fälle in Betracht kommen wird,

- Formularsprachen und graphische Programmierung anhand einer symbolischen Darstellung des automatisierten Prozesses ebenfalls häufig in Frage kommen,

- ALGOL, BASIC und problemorientierte anwendungsspezifische Spezialsprachen zur Zeit noch von geringer Bedeutung sind, letztere aber Zukunftschancen besitzen dürften.

Auch diese Ermittlung ist mit den Vorbehalten, wie sie zum Thema "vermeintliche Wohltat" eingangs erläutert wurden, belastet. Es wurden dem Befragten nämlich nur Programmbausteine vorgelegt, die der Entwerfer wohlwollend für "den" Anwender ausgewählt hatte, ohne das letzterer auf dieses Angebot direkt Einfluß nehmen konnte.

3.4 Wirtschaftlichkeitsstudien

Überwiegend werden Kosten-Nutzen-Erwägungen zum Prozessrechner-Einsatz angestellt, die auf der Bewertung eingesparter Arbeitszeit beruhen. Dieser werden die durch den Prozessrechner-Einsatz verursachten Zusatzkosten gegenübergestellt. Bei der Anschaffung wird der Kauf des Prozessrechners klar gegenüber Miete und Mietkauf bevorzugt.

3.5 Erzielte Verbesserungen durch den Prozessrechner-Einsatz:

- An erster Stelle steht die Verbesserung der Produktqualität, erreicht durch strenge Prozessüberwachung und genaues Einhalten der Spezifikationen,

- an zweiter Stelle die Einsparung der Arbeitszeit und Personal,

- an dritter Stelle die Steigerung der Produktionsausbeute als Folge einer intensiveren Nutzung der Prozess-Anlagen, der Erweiterung der Anlagenverfügbarkeit und der Ausschußverringerung,

- an vierter Stelle steht das Einsparen von Energie, Rohstoffen und Lagerraum, und

- an letzter Stelle die Senkung von Reparatur- und Wartungskosten durch gezielte Langzeitüberwachung der technischen Einrichtungen und schonende Fahrweise der Prozessanlagen.

3.6 Erwartungen des Anwenders

Zum größten Teil haben sich die ursprünglichen Erwartungen bezüglich der Automatisierung durch einen Prozessrechner voll erfüllt, wenngleich von Fall zu Fall die Anwendungssoftware nicht kurzfristig verfügbar war, Vorurteile gegen den Prozessrechnereinsatz sowohl beim Management als auch bei den betroffenen Arbeitnehmern zu überwinden waren und die Software-Unterstützung durch den Lieferanten als nicht ausreichend betrachtet wurde.

Neue Erwartungen der befragten Anwender bezüglich verbesserter Rechnerausstattung und Systembedienung bezogen sich in folgender Reihenfolge auf: Sicherheit und Zuverlässigkeit, schnelle Reaktionszeit, auf größere Arbeitsspeicher, Bildschirmsysteme, Datensicherung, Hardwarekompatibilität, verbesserte Dokumentation, Testmöglichkeiten im On-line Betrieb, Rechnerkopplung, höhere und schnellere Software-Verfügbarkeit und schließlich komfortableres Bit-Handling.

Bei den Erwartungen an eine anwenderorientierte und benutzerfreundliche Prozessprogrammiersprache haben einen hohen Stellenwert: leichte Erlernbarkeit und schnelle Programmierung, Änderungsfreundlichkeit, größere Transparenz, Portabilität und einfache Dokumentation (Selbst-Dokumentation). Die Ansichten über die Einheitlichkeit und Universalität einer zu benutzenden Prozess-Programmiersprache sind jedoch geteilt: zum einen wird

eine universelle Prozessprogrammiersprache wie beispielsweise PEARL / 8 / für zweckmäßig gehalten, zum anderen wird PEARL schon wieder als viel zu mächtig und umfangreich, dagegen Spezialsprachen mit einem Minimum an Befehlen ohne viel "EDV-Ballast" für andwenderfreundlich angesehen. (Siehe auch /6,7/).

Bibliographie

/1/ Proceedings of the IFAC/IFIP Workshop on Real-Time Programming. Ed.by P.D. Griem. Boston, Cambridge, Mass. 21/22. August 1975. Veröffentlicht von der Instrument Society of America, Pittsburg, 1976.

/2/ Dr.J.Witt (SIEMENS): "Grundüberlegungen zu einer anwenderfreundlichen Software-Benutzerschnittstelle". Eigene Notizen zu einem Kolloquiumsvortrag am 29.4.1976 an der Universität Karlsruhe.
Siehe auch: "Prägung durch die Muttersprache". Bericht in der Computerzeitung vom 26.1.1977, Seite 12 ff.

/3/ "Control Engineering Survey: On-line Digital Computers in Control Engineering"- The Magazine of Control and Information Systems. Donelly Publication, New York, 1971.

/4/ G. Friesland, H. Ovenhausen (SCS Hamburg)
"Test von Prozessrechnersoftware - Anwenderumfrage und Literaturanalyse"
PDV-Bericht 39, GfK Karlsruhe, Mai 1975

/5/ R. Hoffmann
"Vergleichende Untersuchung anwendererprobter Prozessrechensysteme (Erfassung, Auswertung, Ergebnisse, Vorschläge"
Diplomarbeit am Lehrstuhl für Planungs- und Programmiertechniken für Prozessrechner (Prof. Rembold) 1976, Institut für Informatik III, Universität Karlsruhe, Postfach 6380.
Muster der verwendeten Fragebögen können vom Lehrstuhl (z.Hd. G. Koch) bezogen werden.

/6/ L. Berscheid
"Untersuchungen über die Anpassung von Prozessprogrammiersprachen an die Anwendungsproblematik"
Diplomarbeit am Lehrstuhl für Planungs- und Programmiertechniken für Prozessrechner (Prof. Rembold), Institut für Informatik III, Universität Karlsruhe, 1976.

/7/ G. Koch
"Stand und Trends der Programmierung von Mikroprozessoren"
ELEKTRONIK 1 und 2, Jan./Feb. 1977, S. 63-66 und S. 66-71

/8/ K. H. Timmesfeld et al.
"PEARL - Ein Vorschlag für eine Prozeß- und Experimentautomationssprache"
GfK Karlsruhe, Bericht KFK-PDV 1 (1973)

Rechnerkopplung Tektronix 4051 - Siemens 330

Klaus Weise

Physikalisch-Technische Bundesanstalt, Braunschweig

Ein BASIC-Programmsystem für den Tischrechner Tektronix 4051 ermöglicht den alphanumerisch-graphischen Terminalbetrieb über eine V24-Schnittstelle zum Prozeßrechner Siemens 330 sowie die Wechselwirkung von Programmen in beiden Rechnern mittels Bedienungsanweisungen. Insbesondere können Bedienungsanweisungsfolgen an die Siemens-Grundsoftware automatisch ablaufen und dadurch die Handhabung des Prozeßrechners 330 wesentlich erleichtern.

1. Einleitung

In der Physikalisch-Technischen Bundesanstalt (PTB) in Braunschweig wird bis 1980 ein Prozeßrechner-Verbundsystem für Meßaufgaben aufgebaut. Es wird aus etwa 20 Rechnern Siemens 330 und einigen Rechnern Digital Equipment PDP 11 bestehen, die miteinander und mit einem Großrechner Telefunken TR 440 gekoppelt werden. Neun Prozeßrechner Siemens 330 laufen bereits ungekoppelt. Das Prozeßrechner-Verbundsystem dient dazu, bis später etwa 70 bis 100 in 15 Gebäuden der PTB verstreute Meßplätze, d.h. viele verschiedenartige physikalische Experimente zu automatisieren,

die Meßwerte gleichzeitig zu erfassen, die oft mit einer hohen Rate von 10^5 bis 10^6/s anfallen, sie zu verarbeiten und die Meßvorgänge zu steuern. 45 Meßplätze werden z.Z. mit Hilfe eines selbstentwickelten Interface-Systems an die Prozeßrechner angeschlossen.

Auf einer unteren Ebene des Rechnersystems der PTB spielt der Tischrechner Tektronix 4051 eine hervorragende Rolle. 15 Stück dieses Tischrechners befinden sich im Einsatz, acht weitere sind bestellt, fünf sind gekoppelt mit den Prozeßrechnern Siemens 330, die übrigen werden in Kürze angeschlossen. Der Rechner Tektronix 4051 wie auch der ähnliche, aber etwas schnellere Tischcomputer Hewlett-Packard HP 9845 sind besonders geeignet als intelligente Prozeß-Terminals am Meßort im technisch-wissenschaftlichen Versuchsbetrieb bei kleineren, langsamen Meßprozessen.

2. Eigenschaften des Tischrechners Tektronix 4051

Bei den Experimentatoren mit geringen Erfahrungen und Ambitionen in der Prozeßdatenverarbeitung ist der Rechner 4051 von Tektronix sehr beliebt. Er ist leicht über eine alphanumerische Tastatur zu bedienen und kann einfach und flexibel in BASIC programmiert werden, wobei besondere Editiertasten nützlich sind. Sehr vorteilhaft sind die Möglichkeiten der Programmierung graphischer Darstellungen, der Textbearbeitung und der Belegung bestimmter Tasten mit programmierten Funktionen des Anwenders. Der Darstellung von Programmen, Daten und Zeichnungen dient ein Speicherbild-

schirm, der Langzeitspeicherung von Daten und Programmen, auch in Segmenten, eine interne Magnetbandkassettenstation. Der Arbeitsspeicher des preisgünstigen Rechners kann von 8 auf 32 kbyte ausgebaut werden. Hardcopy- und Floppy-Disc-Einheiten, Plotter, Drucker und andere Peripheriegeräte sind anschließbar, über einen IEC-Bus auch damit kompatible Meßgeräte und andere mittels eines BCD-Interfaces. Die Meßdatenerfassungsrate liegt zwar nur bei etwa 100/s, doch ist der Rechner wegen seiner Interruptmöglichkeit prozeßgeeignet. Ein Computer-Terminal-Interface (V24-Schnittstelle) vervollständigt den Tischrechner 4051, der damit als alphanumerisch-graphisches Terminal, als Satellitenrechner oder als externe Magnetbandstation eines anderen Computers betrieben werden kann.

3. Hardware der Rechnerkopplung

Für die Kopplung der Rechner Tektronix 4051 und Siemens 330 wird auf der Seite des ersteren das Computer-Terminal-Interface verwendet, dessen Parameter bei der Initialisierung per Programm u.a. auf Vollduplexbetrieb und die maximale Datenübertragungsrate von 2400 bit/s gesetzt werden.

Auf der Seite des Prozeßrechners 330 wird eine auf die gleiche Datenübertragungsrate eingestellte Teletype-Anschaltung mit V24-Schnittstelle benutzt, die von der Firma Siemens für den Anschluß von Tektronix-Terminals der Serie 4000 vorbereitet wurde. Für die

Verbindung der Rechner genügt als Standleitung ein vieradriges Telefonkabel. Im Organisationsprogramm des Prozeßrechners 330 ist der Tischrechner 4051 als Teletype-Blattschreiber TTYE bzw. TTYA mit den Gerätenamen BTAE bzw. BDRA und der jeweiligen Gerätenummer generiert.

4. Software der Rechnerkopplung

Die Software für die Kopplung der Rechner ist äußerst einfach, transparent und erweiterungsfreundlich konzipiert. Für den Prozeßrechner Siemens 330 ist überhaupt keine Ergänzung erforderlich, da der Satellitenrechner 4051 wie ein Ein- und Ausgabe-Blattschreiber angesprochen wird.

Die Struktur der in BASIC programmierten Kopplungssoftware im Tischrechner 4051 ist im Bild 1 dargestellt. Das Programmsystem besteht aus einem kurzen, nur etwa 80 Anweisungen umfassenden, residenten Kopfteil und beliebig vielen Segmenten, jedes in einer File eines Kassettenmagnetbandes. Die Segmente werden erst bei Aufruf der Filenummer nacheinander geladen und durchlaufen. Das System kann durch neue Segmente leicht ergänzt werden.

Der Kopfteil enthält neben dem Programmstück für die Initialisierung einiger weniger Parameter des Softwaresystems und des Terminal-Interfaces vier globale Unterprogramme, die von allen Segmenten über feste Zeilennummern einer Sprungleiste oder durch Drücken bestimmter Anwenderfunktionstasten angesprochen wer-

den können. Das erste Unterprogramm bewirkt den Übergang in den Terminalzustand, in dem der Rechner 4051 wie ein alphanumerisch-graphisches Datensichtgerät oder ein Ein- und Ausgabe-Blattschreiber betrieben werden kann. Der Rücksprung erfolgt beim Betätigen einer Rückkehrtaste. Das zweite Unterprogramm sendet die Bedienungsanweisung /DATA; an den Prozeßrechner Siemens 330, empfängt von dort Datum und Uhrzeit und stellt sie auf dem Bildschirm dar. Diese primitive Funktion dient u.a. zum Überprüfen der Rechnerkoppelstrecke. Das vierte Unterprogramm lädt ein aufgerufenes Segment, durchläuft es und löscht es danach wieder.

Den Kern des gesamten Programmsystems bildet das dritte Unterprogramm. Es vermittelt die Wechselwirkung der Programme in beiden Rechnern durch Übertragung von Zeichenstrings. Es wird versorgt mit einem String, der abgesendet wird und z.B. eine Bedienungsanweisung darstellen kann, sowie mit zwei Strings, die das Ende der zu empfangenden Antwort des Rechners 330 auf den gesendeten String alternativ anzeigen. Je nach Stellung zweier Software-Schalter wird die Antwort auf dem Bildschirm oder auf Magnetband aufgezeichnet. Der Rücksprung erfolgt, wenn einer der Endestrings erkannt wird. Ein Beispiel für die Anwendung des Unterprogramms zeigt Bild 2.

Die komplizierteren Funktionen des Programmsystems sind auf die Segmente verteilt, von denen sieben fertiggestellt sind und die für allgemeine und spezielle Aufgaben auch von wenig erfahrenen Anwendern beliebig

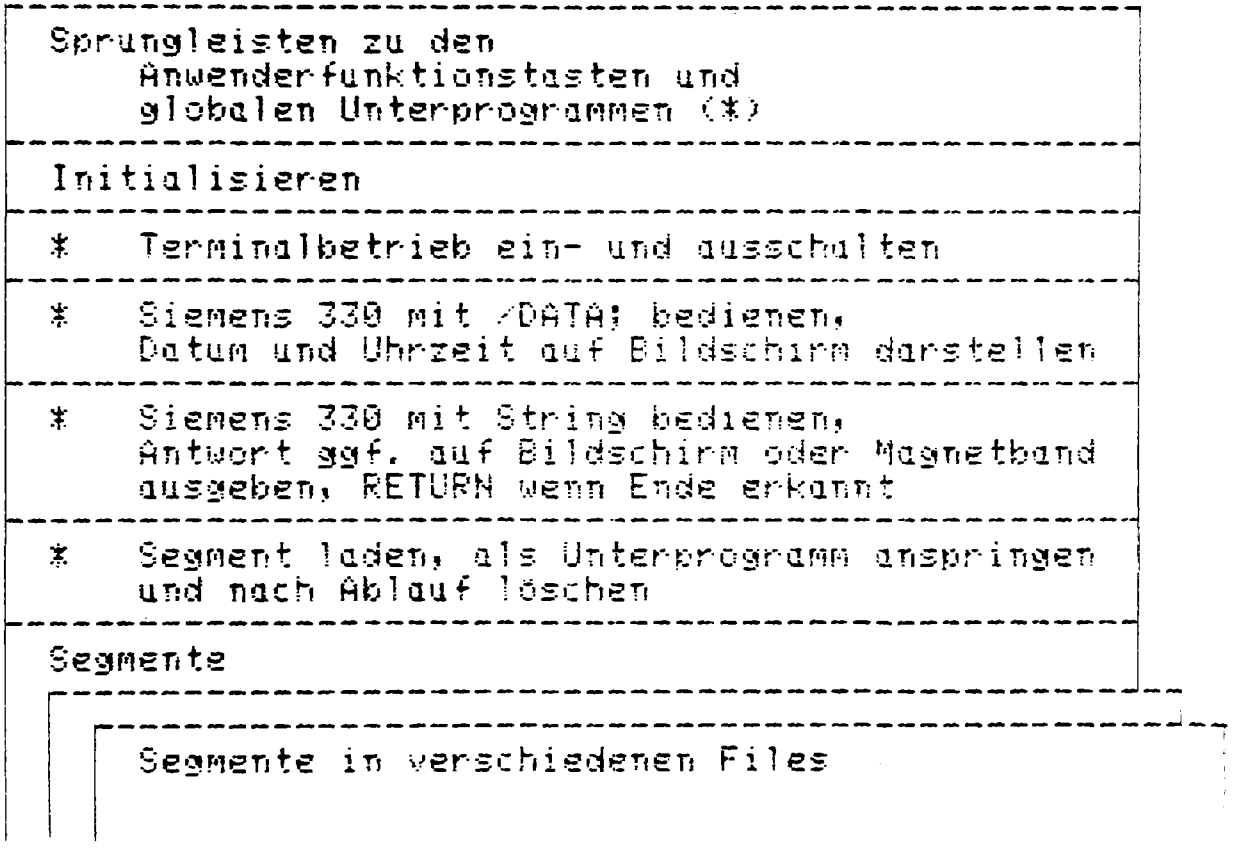
und leicht um weitere ergänzt werden können. Zwei der vorhandenen Segmente laden, falls nötig, in einem Zug die Dienstprogramme DIPOS, SM30, AS30, FC30, BD30, TEPOS und LOEDIT mit dem Common Code CCEDIT, starten sie und bereiten sie für Routineanwendungen vor; oder sie beenden sie und löschen sie eventuell. Zwei weitere Segmente makroübersetzen und assemblieren ein in ASS300 mit Makros geschriebenes Quellprogramm bzw. compilieren und binden ein solches in Prozeß-FORTRAN und laden und starten die erstellten Grundsprache-Programme. Einzelne der nacheinander ablaufenden Arbeitsphasen können übersprungen und z.B. nur eine Syntaxprüfung durchgeführt werden. Die Angabe der beteiligten Elemente reicht aus, da die Bibliotheken voreingestellt sind. Bei einem Schnellmodus genügt der Name eines Quellspracheelements, um mit einem standardisierten Element für die Bindersteuerung ein FORTRAN-Programm zu starten. Diese Segmente sind besonders wichtig für die Anlaufphase der Meßprozeßautomatisierung, in der Programme sehr oft geändert und getestet werden müssen, und erleichtern sehr die recht unhandliche Bedienung des Prozeßrechners 330.

Lediglich für Testzwecke überträgt das fünfte Segment das Buch der FORTRAN-Systembibliothek GSL in eine File eines Kassettenmagnetbandes. Als Programmbeispiel ist dieses kurze Segment im Bild 2 dargestellt. Mit dem sechsten Segment können eine File gelesen oder ein Inhaltsverzeichnis eines Magnetbandes geschrieben werden. Das letzte Segment schließlich dient dazu, von einem Programm im Rechner 330 her die

sehr wirksame interne Graphik des Tischrechners anzusprechen. Je nach den empfangenen Parametern verzweigt das Segment auf die verschiedenen Graphikbefehle, z.B. DRAW. Eine SUBROUTINE in FORTRAN sendet die Parameter und erlaubt durch eine Reihe von ENTRY-Anweisungen die BASIC-analoge Programmierung der Zeichnungen in FORTRAN, z.B. entspricht dem genannten Befehl DRAW ein CALL DRAW. Nachteilig ist die verlangsamte Arbeitsweise.

Wie das Beispiel der Programmierung von Bedienungsanweisungsfolgen an den Prozeßrechner 330 in Bild 2 zeigt, arbeitet das Softwaresystem ähnlich wie das Programm MONITOR, nur werden hier die Kommandosequenzen direkt in der höheren Sprache BASIC geschrieben. Dadurch werden auch Kommandoprogramme mit Schleifen, Verzweigungen, Unterprogrammen möglich, ohne daß ein interpretierendes Steuerprogramm erforderlich wäre. Stringoperationen erlauben die Parametrisierung der Bedienungsanweisungen.

Bild 1: Softwarestruktur der Rechnerkopplung
TEKTRONIX 4051 - SIEMENS 330
(BASIC-Programm im Tektronix 4051)



```
1000 REM TERMINAL : BUCH Bibliothek GSL als Test       23.02.1978
1010 PRINT "BUCH Bibliothek GSL  in File 7"
1020 FIND 7                          Magnetband positionieren
1030 M=1                             Softwareschalter auf Magnetbandausgabe setzen
1040 C$="B:10:"                      Programm-Nr. von DIPOS, B = STX
1050 Q$="DIPOS :"                    1. Endeerkennungsstring
1060 R$=":::"                        2.     "          (hier ohne Bedeutung)
1070 B$=C$&"PR:<BDRA>;C"             Bedienungsanweisung als Sendestring, C = ETX
1080 GOSUB 91                        Globales Unterprogramm für die String-
1090 B$=C$&"G:<PLSK(1),GSL>;C"                 übertragung aufrufen
1100 GOSUB 91
1110 B$=C$&"BUCH:G;C"
1120 GOSUB 91
1130 M=0                             Schalter rücksetzen
1140 RETURN
```

Bild 2: Beispiel für die Programmierung von Bedienungsanweisungsfolgen

Mikroskopbildverarbeitung mit dem Prozeßrechner S330 und dem Programmsystem DIBIVE

P. Gais, K. Rodenacker, W. Abmayr, G. Schwarzkopf
GSF -Institut für Strahlenschutz- 8042 Neuherberg

Im Rahmen des BMFT Projektes TUDAB, d.h. Tumor Diagnose durch automatische Bildanalyse, wird eine große Anzahl von Zellbildern unter dem Mikroskop elektronisch abgetastet und digitalisiert. Aus den Zellbildern werden Merkmale extrahiert, deren Klassifizierung wiederum eine Einteilung der verschiedenen Zellen in Klassen zwischen gutartigen Zellen und Karzinomzellen erlaubt.

In Abbildung 1 ist ein Blockschaltbild des verwendeten Bildaufnahme- und Verarbeitungssystems dargestellt.

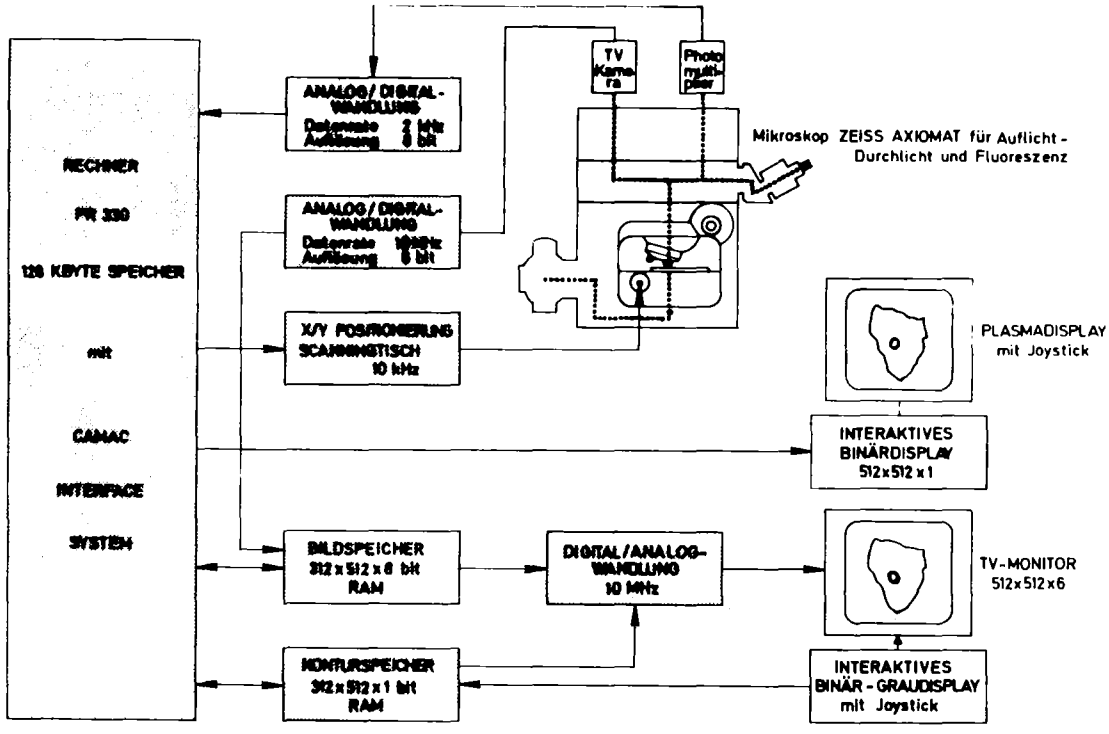


Abb. 1 TV-SMP Bildaufnahme- und Verarbeitungssystem

Das System besteht aus dem Meßmikroskop Axiomat mit zwei Abtasteinrichtungen, einer Fernsehkamera und einer mechanischen Scanningeinrichtung mit Scanningtisch und Photomultiplier. Das Fernsehsystem liefert mit einer Datenrate von 10 MHz Meßwerte, jedoch mit beschränkter Grauauflösung (max. 30 Graustufen). Das Scanningphotometersystem löst bei einer um den Faktor 1.000 geringeren Datenrate bis zu hundert Grauwerte auf und wird vor allem für photometrische Messungen im Zellkern verwendet.

In dieses Meßmikroskop werden die zu untersuchenden Zellen als Abstrichpräparat eingelegt und mit einer örtlichen Auflösung bis zu einem halben /um abgetastet und digitalisiert. Verarbeitungsergebnisse bzw. Darstellungen der abgetasteten Zellbilder werden auf dem Plasma-Display /1/ binär bzw. auf dem TV-Monitor als Graubilder ausgegeben. Die Analog-Digitalwandler, die digitalen Ein- und Ausgaben sowie der Fernsehbildspeicher sind über ein Camac-System an den Rechner S330 gekoppelt. In Abbildung 2 ist eine Aufnahme des Systems im Labor zu sehen.

Abb. 2 Bildanalyselabor

Programmsystem DIBIVE

Zur Steuerung der Bilderfassung zur Bildvor- und Bildverarbeitung sowie zur Bilddarstellung und Archivierung wurde das Programmsystem DIBIVE (Digitale Bildverarbeitung) entwickelt /7/.

DIBIVE wurde in Assembler ASS 300 implementiert und ist durch Fortran- bzw. Assemblerunterprogramme erweiterbar. Insbesondere erlaubt der eingebaute Prozedurinterpreter die Programmierung von komplexen Bildverarbeitungsfunktionen mit Hilfe von Unterprogrammtechnik und Sprungbefehlen. Zur Programmierung von Prozeduren sind Testhilfen, wie Trace, Prozedurablaufprotokoll und Haltepunkte eingebaut. Die Datenströme können über alle Geräte der Standardperipherie geleitet werden.

Im dritten Bild ist ein Blockschaltbild des Programmsystems DIBIVE zu sehen, in der die möglichen Datenströme der Kommandoeingabe, der Kommandointerpretation, der Abarbeitung von Funktionen sowie die Ausgabeaufbereitung dargestellt sind.

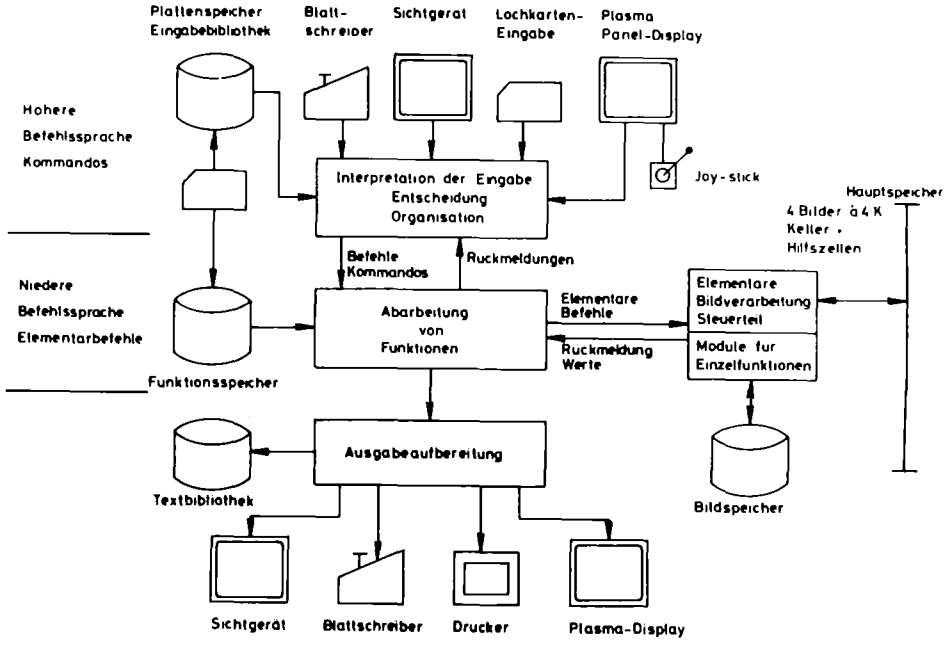


Abb. 3 Programmsystem DIBIVE zur Bildverarbeitung

Das Programm DIBIVE ist segmentiert (overlay) und hat eine Länge von 10 K Worten (16 bit) im Hauptspeicher und ca. 80 K Worten auf Plattenspeicher. Durch Erweiterungen und sinnvolle Anordnung dieser Erweiterungen in Segmenten verlängert sich das Programm nur geringfügig. Zum Ablauf sind noch 16,5 K Worte Datenbereiche und 2,7 K Worte Unterprogramme nötig, die jedoch auch von anderen Programmen benutzt werden. Das Programmsystem ist plattenspeicherresident und belegt nach Start des Ablaufteiles den Laufbereich. Insgesamt werden 8 Objektnummern belegt.

Funktionsbeschreibung

DIBIVE gliedert sich in vier funktionelle Programmteile:

- Vorbereitungsteil
- Kommandointerpreter
- Prozedurinterpreter
- Funktionsteil

Dazu kommen noch Datenbereiche und Systemunterprogramme. Es handelt sich hierbei um Funktionen zur Auswertung von Alarmen der Camac-Peripherie /2/, der Umschlüsselung von Eingaben /3/, der Organisation von Bibliotheken /4/ und der Organisation von digitalisierten Bildern /5/.

Vorbereitungsteil

Nach dem Start des Programms DIBIVE werden über den Vorbereitungsteil die Geräteeinstellungen vorgenommen. Die Einstellungen erfolgen, ähnlich wie in allen Dienstprogrammen, mit:

- EQ : für das Eingabegerät für Kommandos
- AT : für die Ausgaben von Texten
- EB : für die Eingabe von Bildern
- AB : für die Ausgabe von Bildern
- BB : für die Prozedurbibliothek
- EM : Eingabe von Magnetband
- AM : Ausgabe auf Magnetband

Die jeweils eingestellte Gerätekonfiguration kann mit dem Save-Kommando gerettet werden.

Kommandointerpreter

Über das Kommando "RUN" wird nach erfolgter Geräteeinstellung der Ablaufteil gestartet. Dabei werden über das gewählte Eingabegerät Kommandoeingaben angefordert bzw. eingezogen. Alle Kommandos eines Ablaufes bilden zusammen ein Bibliothekselement /6/. Im Dialog wird daher nach Start des Ablaufteiles explizit die Namenskarte angefordert. Erst nach Eingabe dieses Namens erfolgt die Ausgabe der Zeichenfolge "ABF" für Ablaufbefehl. Der in der Elementanfangskarte angegebene Name wird auch als Name für das Ausgabeprotokollelement verwendet.

Jeweils nach der Zeichenfolge "ABF" ist die Eingabe eines neuen Kommandos möglich. In Tabelle 1 sind die speziell für die Bildverarbeitung zugeschnittenen Kommandos dargestellt.

Maximal sind in einem Kommando 10 Parameter zulässig. Neben den Kommandoparametern sind noch zwei weitere Bildparameter möglich, die vier verschiedene Hauptspeicherbereiche bezeichnen, in denen Binärbilder stehen oder abgelegt werden können. Diese Bereiche können weiter noch als Datenspeicher für Elementarfunktionen (z.B. Felder für Fortransubroutinen oder Puffer zwischen Platten- und Hauptspeicher oder Kellerspeicher bei rekursiven Prozeduren) verwendet werden.

Prozedurinterpreter

Jedes Kommando spricht eine Prozedur an, in der Eingabeparameter und der Prozedurablauf festgelegt sind. Alle Prozeduren sind Elemente einer Bibliothek auf Plattenspeicher, die formal denselben Aufbau haben, wie ein Eingabeelement mit den Kommandos. Sie sind kartenweise orientiert und blank komprimiert abgelegt. Sie können mit DIPOS geladen oder mit einem EDITOR direkt eingegeben werden.

Die Prozedursprache ist assemblerähnlich aufgebaut, jedoch ohne mnemotechnische Abkürzungen für die Elementarfunktionen. Der Prozedurinterpreter verarbeitet Karte für Karte der Prozedur. Der sequentielle Ablauf der Prozedur kann durch unbedingte und bedingte Sprünge sowie durch Unterprogramme (geschachtelt bis zur Tiefe 10) durchbrochen werden.

Funktionsteil

Der Funktionsteil des Programmsystems DIBIVE besteht aus allen Elementarfunktionen (wie Bild negieren, Bild kopieren), die in verschiedenen Segmenten ab einer gemeinsamen Segmentierkante abgelegt sind. In einem Segment können verschiedene Elementarfunktionen abgelegt werden, soweit die Länge des Segmentes nicht . 5 K Worte überschreitet. Die Elementarfunktionen sind nach Funktionszusammenhängen in Segmente abgelegt.

In Tabelle 2 ist die Binderliste des Programms DIBIVE mit der Baumstruktur dargestellt.

Verarbeitungsbeispiel Schwellsuche

Zur Illustration der Arbeit mit DIBIVE ist in Tabelle 3 die Prozedur für Schwellsuche und deren Ablauf beschrieben.

Das Kommando SCHWE(A,B,BLNI75) ruft die Prozedur zur Schwellsuche auf. Die Parameter A und B bezeichnen die verwendeten Bildbereiche im Hauptspeicher,der Name BLNI75 bezeichnet eine Graubilddatei auf Plattenspeicher. Diese Graubilddatei soll auf dem TV-Monitor als Graubild dargestellt werden, sowie die Grenzen Untergrund-Zytoplasma und Zytoplasma-Zellkern eingezeichnet werden bzw. verändert werden.

Im vierten Bild ist das Verarbeitungsergebnis dargestellt. Auf der linken Seite das Originalgraubild, auf der rechten Seite das Graubild mit den eingezeichneten Grenzen.

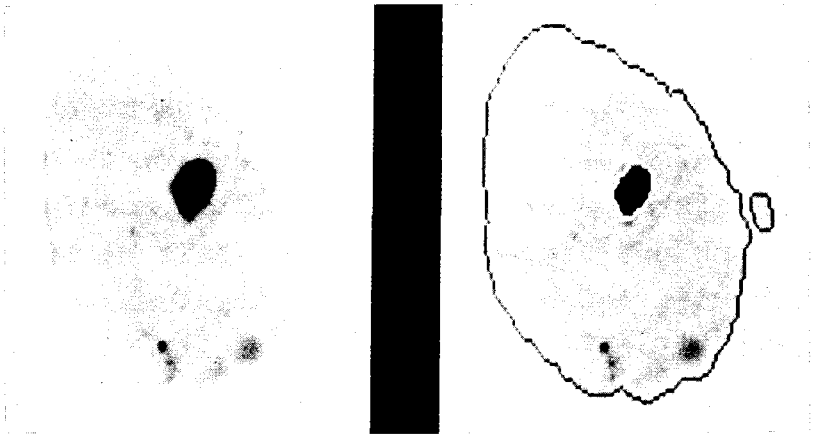


Abb.4

Im fünften Bild ist ein weiteres Verarbeitungsbeispiel zur Bildvorverarbeitung dargestellt. Mit dem Kommando BIVO8 (BLNI75) wird eine Prozedur zur Untergrundelimination und zur Ermittlung von Zytoplasma und Zellkern gestartet.

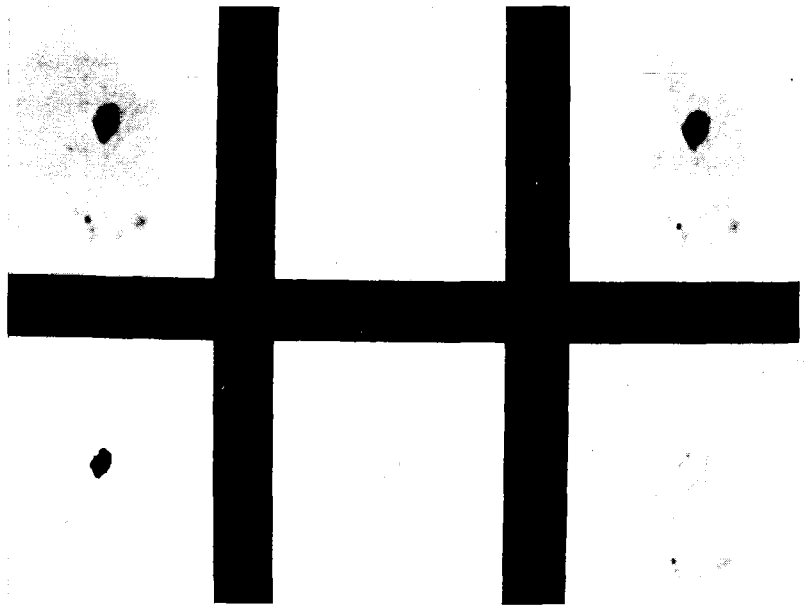


Abb. 5

Mit dem Programmsystem DIBIVE und der eingangs beschriebenen Hardwarekonfiguration steht ein Werkzeug zur Verfügung, das schnell an die verschiedensten Bildverarbeitungsaufgaben angepaßt werden kann und Lösungen in kurzer Zeit bietet. Um dieses Werkzeug noch schneller zu machen, ist daran gedacht, für alle parallelisierbaren Aufgaben der Bildverarbeitung, wie 30 % der Segmentierung, 40 % der Vorverarbeitung und 30 % der Extraktion, zusätzlich noch einen Arrayprocessor zu verwenden, der an den Rechner S330 angeschlossen werden soll. Nach ersten Berechnungen ist damit noch eine Verschnellerung um den Faktor 50 erreichbar.

Literaturverzeichnis

/1/ Eine Camac-Steuerung für graphische Darstellungen auf matrixorientierten Sichtgeräten (Plasma-Displays)
J.Zahn, P.Abend und Z.Komor
Hahn-Meitner-Institut für Kernforschung Berlin GmbH
Bericht Nr. HMI-B 191, Januar 1976

/2/ Beschreibung der Ein- und Ausgabesoftware für CAMAC
Stuckenbrock
Siemens AG, E STE, Karlsruhe

/3/ KOMI, Kommandointerpreter für System 300/16
Siemens AG, Karlsruhe

/4/ BIBEAS, Ein-Ausgabesystem für Bibliotheken
Siemens AG, Karlsruhe

/5/ GBCC, Organisation von digitalisierten Graubildern
K.Rodenacker et.al. i.Vorb.
GSF, Neuherberg

/6/ BISY, Bibliothekssystem
Siemens AG, Karlsruhe

/7/ DIBIVE Programmsystem zur digitalen Bildverarbeitung
K. Rodenacker et.al.
GSF, Neuherberg i.Vorb.

Tabelle 1 Kommandobeschreibung Programm DIBIVE

A) Type, Modewahl, Ende

TYPE	STRING	Einlesen eines Strings aus maxiaml 50 Zeichen
ENDE	-	Beenden des Programmlaufs
PLIS		Prozedurkarten ausgeben

B) Input-Output (binär)

LOAD	PLSK (B)	Laden eines Binärbildes von File
DISP	(B)	Ausgabe eines Bildfeldes auf Display (binär)
STOR	PLSK (B)	Speichern eines Binärbildes auf File

C) Prozeduren zur parallelen Bildverarbeitung (binär)

EXSH	X	(B)	Ergänze X Störstellen horizontal im Bildfeld B
EXSV	X	(B)	Ergänze X Störstellen vertikal im Bildfeld B
EXSP	X	(B)	Ergänze X Störstellen unter +45° im Bildfeld B
EXSN	X	(B)	Ergänze X Störstellen unter -45° im Bildfeld B
LXSH	X	(B)	Lösche X Störstelle horizontal im Bildfeld B
LXSV	X	(B)	Lösche X Störstellen vertikal im Bildfeld B
LXSP	X	(B)	Lösche X Störstellen unter +45° im Bildfeld B
	X	(B)	Lösche X Störstellen unter -45° im Bildfeld B
EHAV	X	(B)	Ergänze X Störstellen horizontal "AND" Ergänze X Störstellen vertikal in B
LHAV	X	(B)	Lösche X Störstellen horizontal "AND" Lösche X Störstellen vertikal in B

D) Boolsche Operationen (binär)

RESE	(B)	Nullsetzen von Bildfeld B
COPY	(B,A)	Copieren von Bildfeld B nach Bildfeld A
AND	(A,B)	log. UND zwischen Bildfeld A und Bildfeld B, Ergebnis in Bildfeld B
OR	(A,B)	log. ODER zwischenBildfeld A und Bildfeld B, Ergebnis in Bildfeld B
EXOR	(A,B)	log. EXKLUSIVODER zwischen Bildfeld A und Bildfeld B, Ergebnis in Bildfeld B
NOT	(B)	Invertierung von Bildfeld B

E) Translation des Bildes (binär)

LINK	X	(B)	Verschieben des Bildfeldes B um X Rasterpunkte nach links
OBEN	X	(B)	Verschieben des Bildfeldes B um X Rasterpunkte nach oben
UNTE	X	(B)	Verschieben des Bildfeldes B um X Rasterpunkte nach unten
RECH	X	(B)	Verschieben des Bildfeldes B um X Rasterpunkte nach rechts

F) Raster, Randlöschen, Objekt löschen (binär)

RAND	IA, IE, JA, JE (B)	Löschen des Bildrandes mit IA, IE, JA, JE in B
ELRA	(B)	Eliminiere Objekte aus B, die den Rand berühren
ELOB	(B) KOX, KOY	Eliminiere das Objekt mit den Koordinaten KOX, KOY in B
MAXOB	(B)	Selektiere das größte Objekt nach B

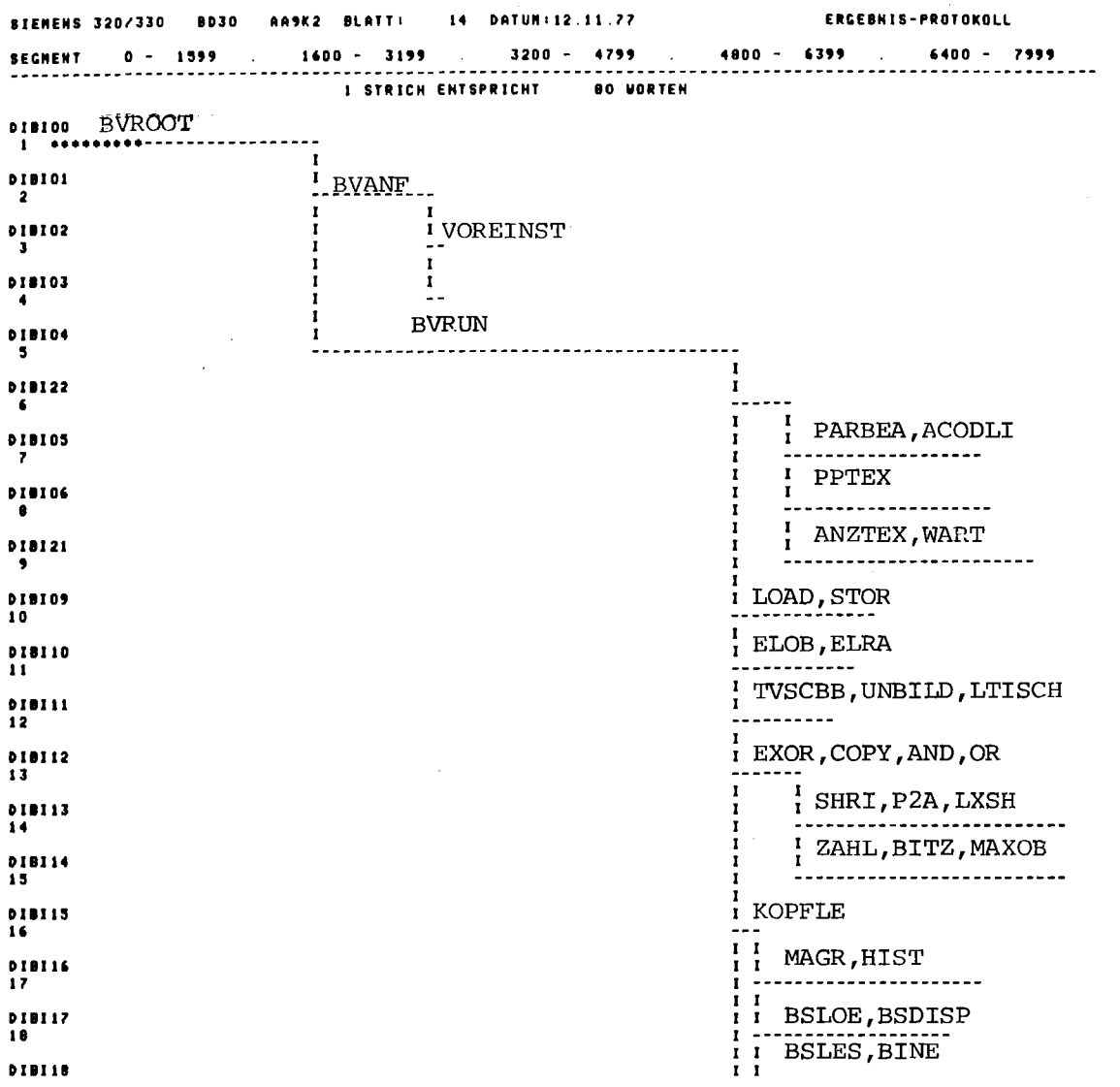
G) Merkmalsextraktion aus Binärbildern

ZAHL	ZAU, ZAO, (B) (ZAA)	Bestimmung der Zahl der Objekte in B, die zwischen ZAU und ZAO liegen.
BITZ	(B)	Bestimmung der Bitzahl in B (BITSI)
P2A	(B)	Bestimmung der Parameter Fläche, Umfang und $U^2/Fx4$) in B
FOR2	(B)	Bestimmung der Parameter SPX, SPY, R1-R4 in B;

H) Prozeduren zur Verarbeitung von Grautonbildern

LEKE	(B)	Erzeugen eines Binärbildes auf Grund der Kernschwelle
LEZY	(B)	Erzeugen eines Binärbildes auf Grund der Zytoplasmaschwelle
LEVEL	LEV1, LEV2 (B)	Erzeugung eines Binärbildes in B aus dem Graubild, das zwischen LEV1 und LEV2 liegt
INSC		Einlesen eines Graubildes und Erzeugung des Histogramms
NORM		Normierung des Histogramms auf das 1.Maximum
HIST	ZAU, ZAO,	Erzeugen des Histogramms und Berechnung der statistischen Größen

Tabelle 2 Binderprotokoll Programm DIBIVE



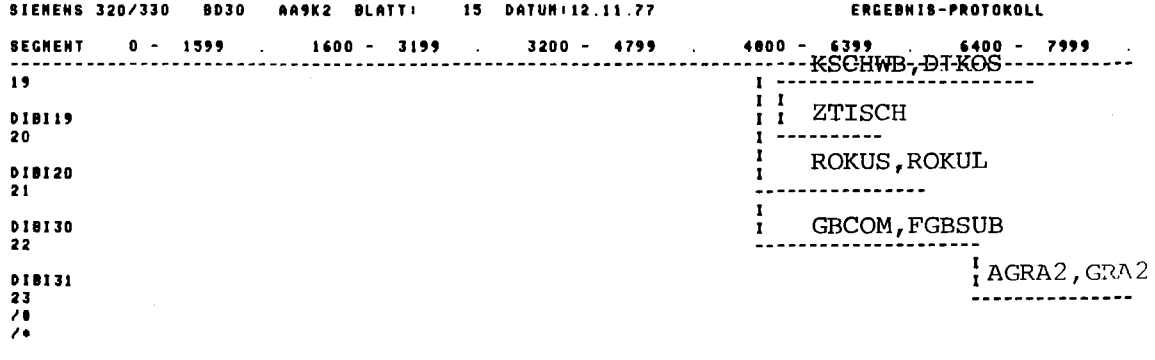


Tabelle 3 Prozedur zur Schwellsuche

```
00002 SCHWE(A,B,BLNI75);                    Kommando zur Schwellensuche
      ------------------------------------------------------------
      BLNI75=OBJEKT PRAEPARAT:54181170      DATUM: 761015  0311
      ------------------------------------------------------------
      SCHWELLE ZYTO:012   SCHWELLE KERN:050
      DIAGNOSE: ????????????????           Ausgabeprotokoll
      A07610150311COMA2205401061O7PA5418117010720163V????????????????BLNI7500000012050
ABF                                         Kopfinformation
  1   /#B075                                SCHWELLKONTOLLE   Prozedur zur Schwellsuche
  2   1 0
  3   E3DATEINAME?                          Eingabe des Parameters "Dateiname"
  4   VE
  5   P64                         #1        Ausgabeprotokoll erzeugen (Unterprogramm)
  6   US27
  7   P67                             BLOE  Bildspeicher loeschen
  8   VA,1 1,B50 2,B50 3          #2
  9   P65                             BLAD  Originalgraubild auf Bildspeicher ausgeben
 10   P64                             LEKE  Erzeugung eines Schwellbildes des Kernes
 11   US40                                        "    des Kernrandes als Binärbild (Unterprogramm)
 12   VA,1 1,B0 2
 13   P61                             MAGR  Weissen Kernrand im Graubild generieren
 14   P63                         #3  LEZY  Erzeugung eines Schwellbildes des Zytoplasmas
 15   US40                                        "    des Zytoplasmarandes (Unterprogramm)
 16   VA,NGGGGGG 1,B255 2
 17   P61                         #4  MAGR  Erzeugung des schwarzen Z.-Randes im GB
 18   VA,B200 2,B50 3
 19   P65                             BLAD  GB mit Rändern aus TV-Display ausgeben
 20   P74                         #5        Warten auf Funktionstaste
 21   BB 10= B1 SP25                        Sprung nach Karte Nr.25 wenn Taste 1
 22   VA,1 1                      #6  KSCHW Bestimmung neuer Schwellen
 23   P72                         #7        Unbedingter Sprung nach Karte Nr. 7
 24   B  SP7
 25   P75                         #8  KOPF  Ausgabe der Kopfinformation
 26   S                                     Ende der Prozedur
 27   P34
 28   A1------------------------------------------------------------
 29   P1
 30   A1      =OBJEKT PRAEPARAT:                DATUM:
 31   AB3 1,1 13,2 16,4 23,5 27                 Unterprogramm für Ausgabeprotokoll
 32   A1------------------------------------------------------------
 33   P42
 34   A1SCHWELLE ZYTO:      SCHWELLE KERN:
 35   AB5 8,6 18
 36   P43
 37   A1DIAGNOSE:
 38   AB6 6,7 9,8 12
 39   UR
 40   VA,B2 1                                   Unterprogramm für Randerzeugung
 41   P25                             EHAV  Ergänzen von Störstellen in h.und v.Richtung der Größe 2
 42   P21                             LXSH  Löschen von Störstellen in horiz.Richtung     "    "  2
 43   P22                             LXSV  Löschen  "      "      " senkr.   "          "    "  2
 44   VA,B1 1
 45   P59                             BLOW  Bild um einen Punkt in jeder Richtung aufweiten
 46   P60                             ELVN  Alle inneren Punkte löschen
 47   P13                             NOT   Bild negieren
 48   UR                                    Unterprogrammrücksprung
 49   /#
```

Funktion von "Schwellsuche"

1 Kopfinformation des angegebenen Graubildes (GB) ausdrucken

2 GB auf Display ausgeben

3 Generierung der Ränder von Zytoplasma und Kern anhand der gegebenenfalls im Kopf angegebenen Grauwertschwellen

4 GB mit schwarzem Rand für Zytoplasma und weißem Rand für Kern auf TV-Display ausgeben

5 Warten auf Eingabe einer Funktionstaste

5.1 Die Schwellen sind richtig, Sprung auf Ende der Prozedur 8

5.2 Die Schwellen müssen eingestellt werden, Sprung auf 6

6 Zur Einstellung der Schwellen am Binär-Plasma-Display und Eintragung der Schwellen in der Kopfinformation steht eine Funktionstastatur zur Verfügung

7 Sprung auf 2 (Ausgabe der zwei Graubilder)

8 Ausgabe der Kopfinformation und Ende der Prozedur

FADABS: Ein Datenbanksystem für den SIEMENS Prozeßrechner 330

F.J. Polster

Kernforschungszentrum Karlsruhe
Institut für Datenverarbeitung in der Technik
7500 Karlsruhe, Postfach 3640
Bundesrepublik Deutschland

Zusammenfassung:

FADABS ist ein Datenbanksystem für Kleinrechner. Die Entwurfsziele sind Datenunabhängigkeit, Flexibilität und Adaptierbarkeit; die Möglichkeiten, die FADABS hierzu bietet, werden dargestellt. Die Benutzerschnittstellen, das heißt die Datendefinitionssprache des Datenbankadministrators und die Datenmanipulationssprache des Anwendungsprogrammierers werden behandelt. Auf die Implementierung wird kurz eingegangen.

E i n l e i t u n g

Am Institut für Datenverarbeitung in der Technik des Kernforschungszentrums Karlsruhe wird gegenwärtig das Datenbanksystem FADABS (Flexibles, Adaptierbares DAtenbank System) auf einem Siemens Prozeßrechner 330 implementiert. FADABS soll als Komponente von verschiedenartigen Systemen eingesetzt werden: Die erste Anwendung - noch in diesem Jahr - stellt das rechnergestützte Splatstoffflußkontrollsystem (Kernmaterialbuchführung, -verfolgung) des KfK dar; FADABS wird weiterhin Bestandteil von Systemen zur on-line Datenerfassung und -verarbeitung in chemischen Labors und bei physikalischen Versuchen sein.

In diesem Bericht soll FADABS aus der Sicht des Anwenders beschrieben werden. Nur soweit zum Verständnis notwendig, wird auf Systemarchitektur und Implementierung eingegangen. In Abschnitt 1 werden die wesentlichen Entwurfskriterien und Eigenschaften von FADABS dargestellt. Abschnitt 2 behandelt ausführlicher die Benutzerschnittstellen. Abschließend folgen in Abschnitt 3 Angaben über den Stand der Arbeiten und Daten zu FADABS.

1. FADABS: Entwurfskriterien, Eigenschaften

1.1 Datenunabhängigkeit

Date /1/ definiert Datenunabhängigkeit als 'immunity of the applications to change in storage structure and access strategy'. Für ein Anwenderprogramm bedeutet dies also zweierlei: es hat keine Kenntnis über

- die Speicherungsformen, d.h. die Art und Weise, in der die Daten auf den Peripherspeichern organisiert sind
- die Zugriffspfade, mittels derer auf die Daten zugegriffen wird (sequentielles Suchen, invertierte Listen, hashing,etc.)

Um ein hohes Maß an Datenunabhängigkeit zu ermöglichen, implementiert FADABS ein relationales Datenmodell; die wesentlichen Begriffe hierbei sind 'Relation', 'Tupel', 'Attribut', die in etwa den gebräuchlicheren Begriffen 'Datei' (file) bzw. 'Satz' (record) bzw. 'Feld' (field) entsprechen.

Eine Relation wird häufig als Tabelle dargestellt:

- eine Zeile entspricht dabei einem Tupel
- eine Spalte enthält die Werte der Tupel der Relation zu einem bestimmten Attribut.

Die Attributwerte eines Attributs sind von einem bestimmten Typ, den Attributtyp ('domain' in /1/). FADABS kennt fünf Attributtypen, diese sind mit den entsprechenden FORTRAN-Datentypen in Tabelle 1 aufgeführt. A*n bedeutet 'Zeichenkette der Länge n'.

Attributtyp	Kompatibler FORTRAN-Datentyp
I*2	INTEGER*2
I*4	INTEGER*4
R*4	REAL
R*8	DOUBLE PRECISION
A*n $o<n$	INTEGER*2 - Feld mit der Dimension $\lceil n/2 \rceil$

Tabelle 1: Die FADABS-Attributtypen

B e i s p i e l:

Die Telefonliste des IDT könnte man als eine FADABS-Relation mit dem Namen TELIDT und den 4 Attributen PHONE, ROOM, NAME und POS implementieren; ein Tupel dieser Relation gibt für einen IDT-Mitarbeiter, dessen Fernsprechnummer (Attribut PHONE, Typ I*2), die Bezeichnung des Raums, in dem er arbeitet (Attribut ROOM, Typ A*5) seinen Namen (Attribut NAME, Typ A*14) und seinen Titel (Attribut POS, Typ A*4) an:

PHONE	ROOM	NAME	POS
4226	103	Borrmann	DI
3736	304	Didic	DI
4221	303	Eggenberger	DINF
3927	211	Friehmelt	DR
.....	.....	.....	.....

Attributnamen

←Tupel

1.2 Flexibilität, Dateitypen

Unter Flexibilität wird hier Flexibilität bezüglich der Speicherungsformen verstanden. Um der Forderung nach Flexibilität gerecht zu werden, bietet FADABS verschiedene Möglichkeiten, sogenannte 'Dateitypen', zur Implementierung einer Relation an. Prinzipiell ist es so möglich, eine Relation als hash-Datei oder geordnete Datei etc. zu realisieren, sofern nur der entsprechende Dateityp in FADABS realisiert ist. Darüber hinaus ist beim Systementwurf darauf geachtet worden, daß man nachträglich - etwa um besonderen Anforderungen gerecht werden zu können - neue Dateitypen ins System leicht inkorporieren kann: Die notwendigen Änderungen sind lokaler Art, d.h. sie betreffen nur eine Programmschicht. Umgekehrt ist es damit ebenso leicht möglich, einen Dateityp aus dem System zu entfernen, wenn dieser nicht benötigt wird.
Änderungen dieser Art haben eine Neugenerierung zur Folge, d.h. es sind Programmodifizierungen an FADABS durchzuführen (vgl. hierzu Abschnitt 1.3).

Gegenwärtig (April 1978) sind 2 Dateitypen vorhanden: Für beide gilt

- daß die Datensätze von fester Länge sind,
- daß invertierte Listen auf jedem Attribut errichtet werden können.

Bei Dateityp 1 werden die Datensätze ungeordnet abgelegt; im anderen Fall sind sie nach dem 'key-Attribut' geordnet, außerdem wird hier gewährleistet, daß die Werte des key-Attributs in der Relation höchstens einmal auftreten (Eindeutigkeit des key-Attributs).

1.3 Adaptierbarkeit

Unter Adaptierbarkeit wird hier die Fähigkeit eines DBS verstanden, sich im Rahmen der im System vorhandenen Möglichkeiten an Änderungen im Benutzerverhalten anzupassen.

Um einen hohen Grad an Adaptierbarkeit zu erreichen, wurde FADABS so entworfen, daß es ohne Anhalten des Betriebs oder gar Neugenerierung (vgl. 1.2) in der Lage ist,

- eine invertierte Liste auf einem Attribut einer Relation zu errichten oder zu löschen,
- eine Relation einzurichten oder zu löschen.

Um von diesen Fähigkeiten Gebrauch machen zu können, ebenso wie aus der Definition von 'Datenunabhängigkeit' (der Anwender hat keine Kenntnis von Zugriffsverfahren), folgt, daß FADABS selbst vor jedem Zugriff ermitteln muß, welche Zugriffspfade zum Zeitpunkt der Anfrage existieren und einen optimalen davon bestimmen muß.

Hierzu ein Beispiel:
Hat der DBA festgestellt, daß im Rahmen eines Auskunftssystems vorwiegend nach Tupeln der Relation TELIDT (Abschnitt 1.1) mit einer bestimmten Telefonnummer gefragt wird, so wird er auf dem Attribut PHONE von TELIDT eine invertierte Liste errichten. Tritt im weiteren Verlauf eine Veränderung im Benutzerverhalten auf, so daß etwa vermehrt nach Tupeln mit einem bestimmten Mitarbeiternamen gesucht wird, so kann dies zu einer längeren (durchschnittlichen) Antwortzeit führen, da in diesen Fällen TELIDT sequentiell durchsucht werden muß. Durch Errichten einer invertierten Liste auf dem Attribut NAME können die Antwortzeiten - ohne Unterbrechung des Betriebs! - verbessert werden. Für den Fall, daß die Anfragen nach Telefonnummern nur noch einen unwesentlichen Anteil aller Anfragen bilden, kann das Löschen der invertierten Liste auf PHONE eine Verbesserung der Effizienz mit sich bringen, da dann diese invertierte Liste von FADABS (bei updates) nicht mehr zu verwalten ist.

1.4 Datensicherheit, Datenschutz

Datensicherheit - Schutz vor Verlust von Daten - und Datenschutz - Schutz vor unberechtigtem Zugriff zu den Daten der Datenbank - sind wesentliche Aufgaben eines jeden Datenbanksystems.

Zur Gewährleistung der Datensicherheit wird eine 'checkpoint'-Technik benutzt: Der DBA oder das Anwenderprogramm kann einen checkpoint setzen, d.h. die Datenbank oder ein Teil davon wird z.B. auf Magnetband gerettet. FADABS hält alle Änderungen an der Datenbasis seit dem letzten checkpoint fest; bei Datenverlust oder -verfälschung durch software- oder hardware-Fehler kann dann ausgehend vom Zustand der Datenbank beim letzten checkpoint und den seitdem vorgenommenen Änderungen der letzte korrekte Zustand hergestellt (und dann der Rechenbetrieb fortgesetzt) werden.

Zum Schutz vor unberechtigtem Zugriff auf FADABS-Dateien durch andere (Dienst-) Programme werden die Möglichkeiten des ORG benutzt: Benutzerkennzeichen, Eigentümerkennzeichen /3/.

Zur Kontrolle des Zugriffs über die FADABS-Schnittstellen erlaubt der Systementwurf die Realisierung selbst von Verfahren, die den Zugriff auf ein Tupel in Abhängigkeit vom Wert eines Attributes regeln, so daß z.B. einem Anwender (-programm) der Zugriff nur auf solche Tupel einer Mitarbeiterrelation verwehrt werden kann, bei denen das Attribut GEHALT einen Wert größer als z.B. 3000.00 hat.

1.5 Zur Implementierung von FADABS

Das DBS FADABS besteht aus mehreren selbständigen Programmen: Dem FADABS-Nucleus und FADABS-Dienstprogrammen wie DBA-Programm, FORDAM-Prozessor (s. Abschnitt 2.2), interaktives Anfragesystem, Report-Generator, etc.; die Programmierung erfolgt weitestgehend in FORTRAN. (Ausnahme: Die Schnittstelle des Nucleus zum ORG.) Der Nucleus kann von mehreren gleichzeitig ablauffähigen Anwenderprogrammen angesprochen werden, wobei die FADABS-Dienstprogramme ebenfalls zu den Anwenderprogrammen gezählt werden, Bild 1. Alle Anwenderprogramme benutzten als einheitliche Schnittstelle zum Nucleus EDBL (Elementary Data Base Language):

EDBL besteht aus subroutine calls der Form

```
CALL  FADABS (P1, P2, P3, P4).
```

Der Datenaustausch erfolgt über einen dem Nucleus und allen Anwendern gemeinsamen Pufferbereich im Hauptspeicher.

Im folgenden wird der FADABS-Nucleus, wo dieses zu keinem Mißverständnis führt, auch kurz nur FADABS genannt.

Da der vom FORTRAN-Compiler FC30 erzeugte Code nicht reentrant ist, wird FADABS den Anwendern zur Bearbeitung eines Auftrags wie ein Betriebsmittel im Sinne von Betriebssystemen exklusiv zugeteilt; die Koordinierung der Kommunikation mit FADABS erfolgt anhand von Koordinierungszählern des ORG, die Warteschlangeneintragungen erfolgen nach Priorität /3/.

Diese Koordination der Kommunikation mit FADABS ist u.a. Aufgabe des 'FADABS-Laufzeitsystems', ein Modul, der zu jedem Programm hinzuzubinden ist, das FADABS anspricht.

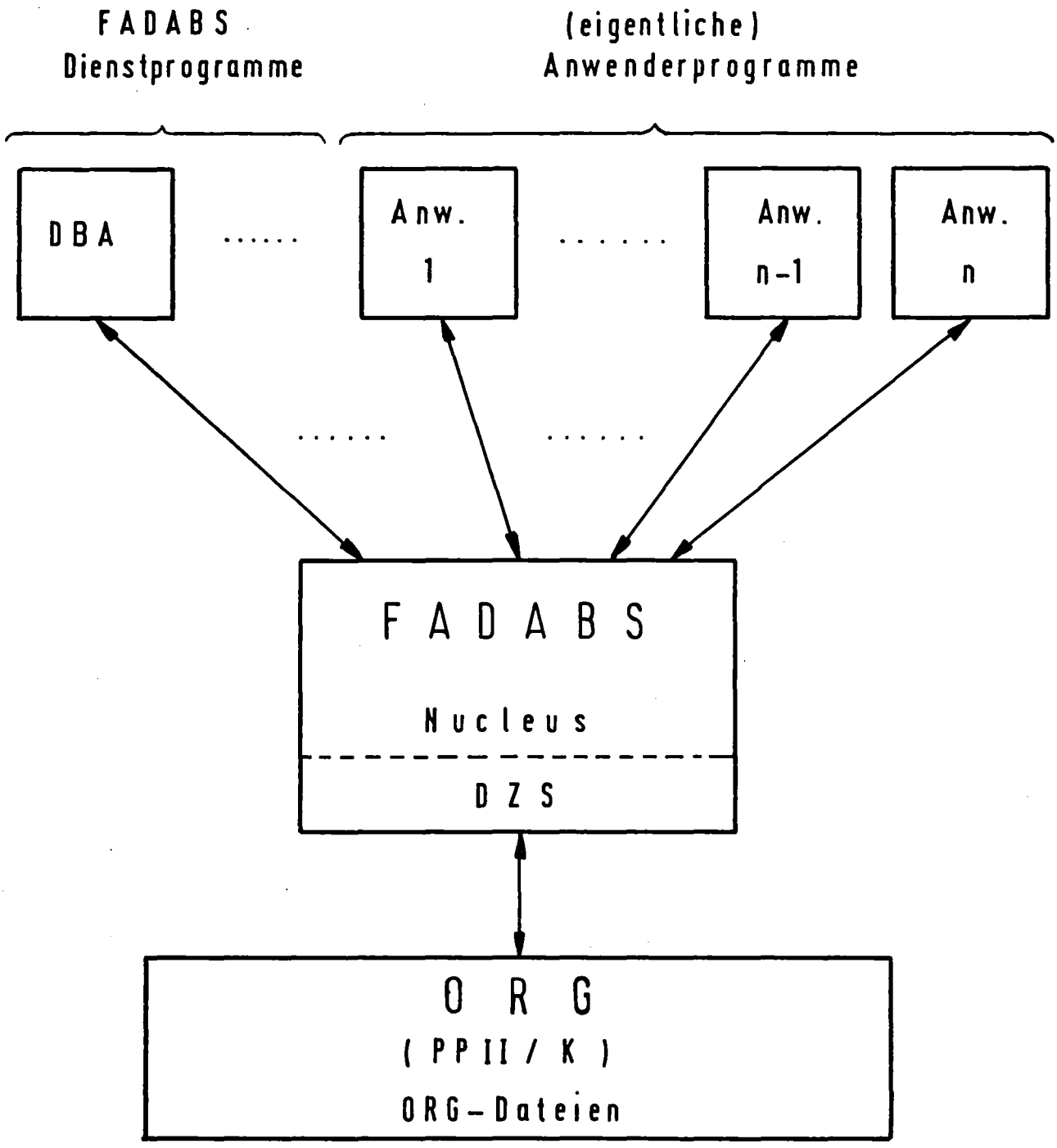


Bild 1: FADABS-Programmstruktur

FADABS benutzt die dateiorganisatorischen Funktionen des ORG 330. Jede Relation wird intern von FADABS in eine blockstrukturierte Datei abgebildet, also Dateien, die aus Blöcken (Seiten, pages) bestehen, in denen im allgemeinen die Daten von mehreren Tupeln abgelegt sind. Die Programmschicht DZS bildet diese FADABS-interne Schnittstelle auf die Strukturen und Operationen des ORG ab; beim E/A-Verkehr mit den ORG-Peripherspeicherdateien wird nach dem LRU-Algorithmus verfahren. Dieser Modul wird näher in /2/ beschrieben.

2. Die FADABS Benutzerschnittstellen

FADABS kennt zwei Benutzerklassen: den Datenbankadministrator (DBA) und den Anwendungsprogrammierer. Die Schnittstellen dieser beiden Benutzerklassen - die Datendefinitionssprache (DDL) des DBA, die Datenmanipulationssprache FORDAM des Anwendungsprogrammierers - sollen hier kurz umrissen werden.

2.1 Die DDL des DBA

Der DBA ist verantwortlich für das ordnungsgemäße und effiziente Arbeiten des Datenbanksystems (vgl. /1/). Ihm sind u.a. gerade die Funktionen zugeordnet, die es den Anwendungsprogrammierern erlauben, datenunabhängige Programme zu schreiben. Bemerkung: Bei einem Kleinrechnersystem wie FADABS wird der DBA und Anwendungsprogrammierer oft dieselbe Person sein, was jedoch nicht wesentlich ist; entscheidend ist die Trennung zwischen den Funktionen des DBA und den Möglichkeiten des Anwendungsprogrammierers.

Die FADABS DDL ist eine einfache Kommandosprache, die Kommunikation mit FADABS verläuft in Form eines Dialogs. Die wichtigsten Funktionen eines FADABS DBA (vgl. hierzu /1/, S. 19-21):

- Einrichten (Löschen) von Relationen
  Zum einen wird hiermit der Aufbau und Inhalt der Datenbasis durch den DBA festgelegt, zum anderen aber auch der Dateityp (d.h. die Speicherungsform) spezifiziert, nach dem die Relation implementiert sein soll.

- Einrichten/Löschen von Zugriffspfaden
  Der DBA hat die Aufgabe, dem Benutzerverhalten entsprechend für effiziente Zugriffsmöglichkeiten zu sorgen. Hierzu stellt FADABS als weiteres Mittel (neben der Bestimmung des Dateityps beim Einrichten einer Relation) die Funktionen 'invertierte Liste errichten (bzw. löschen)' dem DBA zur Verfügung.
- Weitere Funktionen zur Gewährleistung des ordnungsgemäßen Betriebs, z.B.: Setzen von checkpoints, d.h. retten von Relationen, Wiederherstellen des letzten korrekten Zustands einer Relation bei Datenverlust oder Verlust der Datenintegrität im allgemeinen.

Zur Verdeutlichung ist in Bild 2 als Beispiel der DBA-Dialog zum Einrichten der Relation TELIDT von Abschnitt 1.1 dargestellt:

Der Dialog wird durch das Kommando CREL angestoßen; es werden zuerst der Name und der Dateityp der einzurichtenden Relation angefordert. Als nächstes sind die Attribute mit Attributtyp einzugeben. Zur Bestimmung der Größe der ORG-Datei wird nach der Zahl der Tupel gefragt, die die Relation (maximal) aufnehmen soll. FADABS gibt dann die Zahl der Worte aus, die ein Tupel belegt und fordert den sogenannten Blokkungsfaktor an, d.h. die Zahl der Tupel, die mit 1 Plattenzugriff transferiert werden sollen. Schließlich ist noch anzugeben, wo die ORG-Datei physikalisch liegen soll ('PLSK20' bedeutet 'PLSK(2,0)').

Die eingegebenen Daten werden eventuell modifiziert zur Quittierung nochmals vorgelegt und anschließend durch FADABS die Relation eingerichtet.

2.2 Die Datenmanipulationssprache FORDAM

FORDAM (FORTRAN DAta Manipulation) bildet die FADABS-Schnittstelle zum Anwendungsprogrammierer und bietet Funktionen zum Lesen, Löschen und Einfügen von Tupeln aus bzw. in bestehende FADABS-Relationen.

FORDAM ist eine Erweiterung von FORTRAN und besteht aus

- DEFINE-Anweisungen zur Deklaration von (nicht-FORTRAN) Objekten der Typen QUALIFICATION, QSS, TUPLE;
- Operatoren (oder auch 'ausführbaren Anweisungen') zum Lesen aus und Ändern von Relationen;

FORDAM-Anweisungen können an jeder Stelle im Programm benutzt werden, jede Programmzeile, die eine FORDAM-Anweisung enthält, beginnt (in den Spalten 1 und 2) mit '**'.

```
CREL
NAME DER RELATION :
AAAAAA
TELIDT
TYP DER RELATION :
I
1

NAME UND TYP DES  1. ATTRIBUTS :  ?
AAAAAA   A*I
PHONE    I*2

NAME UND TYP DES  2. ATTRIBUTS :  ?
AAAAAA   A*I
ROOM     A*5

NAME UND TYP DES  3. ATTRIBUTS :  ?
AAAAAA   A*I
NAME     A*14

NAME UND TYP DES  4. ATTRIBUTS :  ?
AAAAAA   A*I
POS      A*4

NAME UND TYP DES  5. ATTRIBUTS :  ?
AAAAAA   A*I
;;
ANZAHL DER TUPEL : ?
IIII
0200
TUPELLAENGE :  13 WORTE
BLOCKUNGSFAKTOR : ?
II
10
LOG. GERAETENAME : ?
AAAAII
PLSK20

      DATEN DER RELATION TELIDT

TYP DER RELATION ..... :       1
TUPEL - LAENGE ....... :      13
TUPEL - ANZAHL ....... :     208
BLOCKUNGSFAKTOR ...... :      13
PAGE - LAENGE ........ :     192
PAGE - ANZAHL ........ :      16
LOG. GERAETENAME ..... : PLSK20

ATTRIBUTE   NAME     TYP
            PHONE    I* 2
            ROOM     A* 5
            NAME     A*14
            POS      A* 4

EINGABE - DATEN OK ?    JA/NEIN
JA
RELATION TELIDT EINGERICHTET
:71:  DBA - READY
```

BILD 2: Der DBA-Dialog zum Einrichten der Relation TELIDT
(DBA-Eingaben sind unterstrichen)

An dieser Stelle kann FORDAM nur im Rahmen eines Beispiels erläutert werden; für Einzelheiten sei auf die Sprachbeschreibung verwiesen, der Anhang enthält eine formale Beschreibung der FORDAM-Syntax.

Bild 3 gibt in Auszügen ein Programm wieder, das Anfragen bezüglich der Relation TELIDT von Abschnitt 1.1 durchführt, dieses soll hier kurz erläutert werden:

Um Tupel lesen zu können, sind diese zunächst zu beschreiben, hierzu dienen Qualifikationen: Mit einer DEFINE QUALIFICATION-Anweisung wird eine Qualifikation, d.h. die Beschreibung der gesuchten Tupel, mit einem Namen versehen, z.B. wird in (2) die Beschreibung

> "alle Tupel aus TELIDT, deren Attributwert für PHONE
> gleich dem Wert der Programmvariablen NR1A ist"

mit dem Namen Q1A verknüpft.

Qualifikationen können beliebig komplex sein: beliebig viele Prädikate können verwendet werden, Klammerung ist erlaubt; es sei hierzu auf die Grammatik im Anhang verwiesen. An Stelle von Programmvariablen bei Prädikaten können natürlich auch Konstante spezifiziert werden.

Mit der ASSIGN-Anweisung werden durch FADABS Qualifikationen ausgewertet: In (4) werden die Tupel bestimmt, die zum Zeitpunkt des Aufrufs - also nachdem NR1A ein Wert zugewiesen worden ist - der mit Q1A bezeichneten Qualifikation genügen. In der QSS-Variablen QSSTEL - diese wurde in der DEFINE QSS-Anweisung (1) deklariert - wird dem Programm der von FADABS generierte Name für die Menge der sich qualifizierenden Tupel übergeben.

Mit einer RETRIEVE-Anweisung kann auf diese Menge zugegriffen werden im Sinne einer 'get-next'-Operation, d.h. es wird stets nur 1 Tupel gelesen.

Die RETRIEVE-Anweisung (5) besagt, daß ein Tupel aus der mit QSSTEL bezeichneten Menge verlangt wird und die Attributwerte dieses Tupels gemäß der Tupelvariablen TELNR den Programmvariablen VTEL, VRAUM, VNAME, VPOS zuzuweisen sind. Diese Verknüpfung zwischen Attributen einer Relation und Variablen des Anwenderprogramms ist die Funktion von Tupelvariablen: Die DEFINE TUPLE-Anweisung (3) besagt, daß mit dem Namen TELNR die Vorschrift zu verknüpfen ist, daß der Attributwert von PHONE in der Programmvariablen VTEL abzulegen ist, der von ROOM in der Programmvariablen VRAUM, etc. .

```
      - - - - - - - - - - - - - - - - - - -
**    DEFINE QSS QSSTEL (1)
C
C     DEKLARATION DER QUALIFIKATIONEN
C
      - - - - - - - - - - - - - - - - - - -
**    DEFINE QUALIFICATION Q1A/TELIDT, PHONE .EQ. NR1A/ (2)

      - - - - - - - - - - - - - - - - - - -
**    DEFINE QUALIFICATION Q2A/TELIDT,
**           PHONE .EQ. NR2A   .OR.   ROOM .EQ. RAUM2A /
**    DEFINE QUALIFICATION Q2B/TELIDT,
**           ROOM  .GE. R2B1   .AND.  ROOM .LE. R2B2   /

      - - - - - - - - - - - - - - - - - - -
C     DEKLARATION DER TUPEL-VARIABLEN
**    DEFINE TUPLE TELNR                          }
**                 /TELIDT,                       }
**                 (PHONE ,VTEL  ),               }  (3)
**                 (ROOM  ,VRAUM ),               }
**                 (NAME  ,VNAME ),               }
**                 (POS   ,VPOS  )/               }

      - - - - - - - - - - - - - - - - - - -
C     ANFRAGE Q1A
      WRITE(OUT,10005)
      READ ( IN,10006)  NR1A
**    ASSIGN Q1A TO QSSTEL (4)

      - - - - - - - - - - - - - - - - - - -
C     NAECHSTES TUPEL AUS QSSTEL LESEN,
C                       ZUORDNUNG DER ATTRIBUTWERTE GEMAESS TELNR
**    RETRIEVE TELNR FROM QSSTEL; ON EMPTY QSS GO TO 290 (5)
      WRITE(OUT,10011) VTEL,VRAUM,VNAME,VPOS

      - - - - - - - - - - - - - - - - - - -
```

Bild 3: FORDAM-Anweisungen in einem FORTRAN-Programm (auf die Markierungen wird zur Erläuterung im Text Bezug genommen)

Nach der RETRIEVE-Anweisung können die Attribute in den jeweiligen Programmvariablen wie gewohnt weiter verarbeitet, z.B. wie in Bild 3 ausgegeben werden.

Es sei hier nochmals ausdrücklich darauf hingewiesen, daß der Programmierer sich nicht mit Zugriffspfaden, logischen Gerätenamen des ORG o. ä. beschäftigt und FORDAM so ein hohes Maß an Datenunabhängigkeit bietet. Der Anwender kann sich also ganz auf sein eigentliches Problem konzentrieren und geht nicht auf Probleme der Datenhaltung ein.

Abschließend sei noch bemerkt, daß die FORDAM-Anweisungen eines Programms vor der Compilierung durch einen FORDAM-Prozessor in FORTRAN-Code (genauer EDBL-Anweisungen) zu übersetzen sind. Mit dieser Art der Implementierung von FORDAM wird versucht, den besonderen Verhältnissen bei Kleinrechnern gerecht zu werden /4/.

3. Stand der Arbeiten, Ausblick

Eine erste Version von FADABS ist fertiggestellt (April 1978):

- FORDAM-Schnittstelle, -Prozessor
- DBA-Modul
- FADABS-Nucleus mit 2 Dateitypen

Bis Mitte 1978 werden als weitere Zugriffsmöglichkeiten Invertierte Listen bereitgestellt. Daran schließen sich die Arbeiten zur Datensicherung an. Ein Reportgenerator ist in Arbeit.

FADABS benötigt etwa 20 kW HSP, läuft unter dem ORG 330 (K/PPII) und erfordert mindestens eine Platteneinheit. Das FADABS-Laufzeitsystem umfaßt in der FORTRAN-Version ca. 370 Worte. Die Dienstprogramme zur Datensicherung setzen eine Magnetbandeinheit voraus.

A n h a n g: Die Syntax von FORDAM

1. Erlaeuterungen

Zur formalen Beschreibung der Syntax der FORDAM-Anweisungen wird eine BNF-aehnliche Notation verwendet:

- Die Metasymbole | und ::= haben die uebliche Bedeutung
- Zur Bezeichnung von Nicht-Terminalen werden Kleinbuchstaben und das Minus-Zeichen '-' benutzt.
- Das Zeichen '_' symbolisiert eine Folge von beliebig vielen, mindesten aber einem Leerzeichen.
- Ueberall wo '_' nicht explizit aufgefuehrt ist, koennen (etwa zur Verbesserung der Lesbarkeit bzw. Uebersichtlichkeit) zwischen Terminale ein oder mehrere Leerzeichen eingefueqt werden.
- Der einfacheren Beschreibung von Iterationen dienen die Metasymbole % und %n,m, wobei n und m natuerliche Zeichen repraesentieren:
  % string %n,m bedeutet, dass string mindestens n-mal und hoechstens m-mal auftritt.

Die Nicht-Terminale name, constant, statement sind nicht formal definiert:

name:
Variablenname gemaess ANS-FORTRAN

constant:
Konstante vom Typ INTEGER, REAL oder DOUBLE PRECISION nach ANS-FORTRAN oder CHARACTER-Konstante nach 'draft proposed ANS-FORTRAN' (d.h. eine Zeichenkette, die durch ' begrenzt ist).

statement:
Anweisungsnummer ('statement number') gemaess ANS-FORTRAN

2. Die Grammatik

```
fordam-statement ::=   def-statement
                     | rel-statement
                     | qss-statement
                     | update-statement
                     | retr-statement
                     | on-statement

   def-statement ::=   def-t-var
                     | def-q-var
                     | def-qss-var
       def-t-var ::=   DEFINE _ TUPLE _ t-name / rel-name ,
                       attr-association / FOR _ mode
attr-association ::=   ( attr-name , var-name )
                       %  , ( attr-name , var-name ) %0,n
     def-qss-var ::=   DEFINE _ QSS _ qss-name
       def-q-var ::=   DEFINE _ QUALIFICATION _ q-name
                       / rel-name  ,  qualification /
   qualification ::=   q
                     | *
               q ::=   q-term
                     | q-term .OR. q
          q-term ::=   q-factor
                     | q-factor .AND. q-term
        q-factor ::=   predicate
                     | ( q )
       predicate ::=   attr-name rop value
             rop ::=   .LT.
                     | .LE.
                     | .EQ.
                     | .GE.
                     | .GT.
                     | .NE.
           value ::=   var-name
                     | constant

   rel-statement ::=   open
                     | close
                     | save
            open ::=   OPEN _ rel-name _ FOR _ mode
           close ::=   CLOSE _ rel-name
            save ::=   SAVE _ rel-name

   qss-statement ::=   assign
                     | drop
          assign ::=   ASSIGN _ q-name _ TO _ qss-name
            drop ::=   DROP _ qss-name
```

```
update-statement ::=   delete
                     | insert
                     | replace
         delete ::=   DELETE _ ALL _ TUPLES _ OF _ qss-name
                     | DELETE _ CURRENT _ TUPLE _ OF
                       _ qss-name ; on-empty-qss
         insert ::=   INSERT _ t-name
        replace ::=   REPLACE _ CURRENT _ TUPLE _ OF
                       _ qss-name _ BY _ t-name


  retr-statement ::=   RETRIEVE _ t-name _ FROM
                       _ qss-name ; on-empty-qss
    on-empty-qss ::=   ON _ EMPTY _ QSS _ GO _ TO _ statement


    on-statement ::=   ON _ ERROR _ GO _ TO _ statement


            mode ::=   UPDATES
                     | RETRIEVAL

        rel-name ::=   name
       attr-name ::=   name
        var-name ::=   name
          t-name ::=   name
        qss-name ::=   name
          q-name ::=   name
```

L i t e r a t u r:

/1/ Date, C.J.:
An Introduction to Database Systems
Second edition
Addison Wesley Publishing Company, 1977

/2/ Kerpe, R.:
Eine Implementierung des 'Least Recently Used'
- Seitenersetzungsalgorithmus für den SIEMENS Prozeß-rechner 330
In: Voges, U.:
Tagungsbericht der 9. Jahrestagung des Siemens Prozeß-rechner Anwenderkreises I,
Kernforschungszentrum Karlsruhe, 5.-7. April 1978,
Kernforschungszentrum Karlsruhe: KfK 2642, S. 245-254

/3/ ORG 330K
Beschreibung P71100-B0330-X-X-35
Siemens AG.

/4/ Polster, F.-J.:
Using A Preprocessor To Implement A Data Manipulation Language For A Minicomputer Data Base System
Proceedings of the First Annual Symposium on Small Systems, New York, August 2-3, 1978
ACM SIGMINI

Eine Implementierung des 'Least Recently Used' - Seitenersetzungsalgorithmus für den Prozeßrechner 330

R. Kerpe
Kernforschungszentrum Karlsruhe
Institut für Datenverarbeitung in der Technik
Postfach 3640, 7500 Karlsruhe
Bundesrepublik Deutschland

Zusammenfassung:

Es wird ein System vorgestellt, das es erlaubt, von FORTRAN-Programmen aus ORG-Dateien einzurichten, zu löschen und auf diese Dateien lesend und schreibend zuzugreifen.

Devicelisten werden nicht benötigt.

Beim E/A-Verkehr wird nach dem LRU-Algorithmus verfahren: Ein Plattenzugriff erfolgt nur dann, wenn der angeforderte Datensatz sich nicht im Hauptspeicher befindet.

Einleitung

Wie in /1/ hingewiesen, existiert im Datenbanksystem FADABS ein Modul, der die Schnittstelle zum ORG /2/ darstellt.

Dieser Baustein, DZS (Dateizugriffssystem) genannt, realisiert als selbständige Schnittstelle ORG-Dateiverwaltungsfunktionen als FORTRAN-Call-Aufrufe.

Aus den bekannten Effizienzgründen wird dabei der E/A-Verkehr mit den ORG-Dateien nach dem LRU-Algorithmus abgewickelt.

Das DZS ist als selbständiger Modul implementiert und wird im folgenden beschrieben. In Abschnitt 1 werden die Eigenschaften und Fähigkeiten und in den Abschnitten 2 und 3 die DZS-Aufrufe und Anwendungen dargestellt.

1. Eigenschaften und Fähigkeiten

1.1 Dateiverwaltungsfunktionen

Das DZS führt die folgenden dateiorganisatorischen Funktionen des ORG dem FORTRAN-Anwender zu:

- ORG-Datei einrichten / löschen
- ORG-Datei eröffnen / schließen
- Satz lesen / schreiben.

Mit dem DZS können nur Dateien angesprochen werden, die durch das DZS eingerichtet wurden; wesentlich dabei ist, daß der DZS-Anwender für diese Dateien keine Devicelisten benötigt.

1.2 Der LRU-Seitenersetzungsalgorithmus im DZS

FADABS benutzt blockstrukturierte Dateien, d.h. diese Dateien bestehen aus Blöcken, auch Seiten genannt (engl. page), auf die über eine (Block-) Nummer zugegriffen wird.

Für Seiten ist im DZS ein HSP-Bereich AREA vorhanden, in dem Seiten abgelegt und so lange wie möglich präsent gehalten werden. Werden vom DZS-Anwender Seiten angefordert, die sich noch nicht AREA befinden und AREA bereits voll belegt ist, so wird die am längsten nicht mehr angesprochene Seite aus AREA entfernt und der freigewordene Platz mit dem Inhalt der angeforderten Seite überschrieben.

Dieser Vorgang soll im folgenden näher beschrieben werden.

1.2.1 Bei jedem GET-Aufruf, d.h. das aufrufende Programm fordert eine Seite an, überprüft das DZS, ob die angeforderte Seite sich bereits in AREA befindet.

Falls ja: Das DZS übergibt dem aufrufenden Programm die angeforderte Seite; dazu ist kein Peripherspeicherzugriff notwendig.

Falls nein: Die angeforderte Seite ist also vom Peripherspeicher nach AREA zu transferieren; dabei sind zwei Fälle zu unterscheiden.

Fall 1: In AREA ist Platz vorhanden. Der Transfer wird ausgeführt und die Seite dem aufrufenden Programm übergeben.

Fall 2: In AREA ist kein Platz vorhanden. Die am längsten nicht mehr angesprochene Seite wird aus AREA entfernt, d.h. eine veränderte Seite wird dabei in die entsprechende Datei zurücktransferiert und der belegte Speicherplatz freigegeben, bei unveränderten Seiten wird der Speicherplatz sofort freigegeben.

Veränderte Seiten sind Seiten,auf die mindestens ein PUT-Aufruf abgesetzt wurde, unveränderte Seiten sind Seiten, die nur durch GET-Aufruf(e) angesprochen wurden.

Nach dem Entfernen der am längsten nicht mehr angesprochenen Seite(n), wird beim Transfer der angeforderten Seite wie im Fall 1 verfahren.

1.2.2 Mit jedem PUT-Aufruf wird eine Seite vom aufrufenden Programm nach AREA transferiert.

Befand sich eine Seite mit der angesprochenen Seitennummer bereits in AREA, so wird diese mit dem neuen Seiteninhalt überschrieben; ein Peripherspeicherzugriff findet nicht statt.

Im Konfliktfall, d.h. in AREA ist kein Platz vorhanden und die angesprochene Seite befindet sich noch nicht in AREA, wird die am längsten nicht mehr angesprochene Seite nach dem wie oben beschriebenen Algorithmus entfernt und der Transfer 'aufrufendes Programm →AREA' durchgeführt.

2. DZS-Aufrufe

In diesem Abschnitt werden die DZS-Aufrufe erläutert. Der Hinweis auf die entsprechende dateiorganisatorische Funktion des ORG erfolgt durch Angabe des ORG-Makros.

Dateien werden durch Dateinamen identifiziert; der Parameter FILE repräsentiert bei den folgenden DZS-Aufrufen den "DZS-Dateinamen". FILE ist ein DZS-spezifischer Dateiname.

Durch Übergabe eines Return-Codes im Parameter RET, teilt DZS dem Anwenderprogramm mit, in welcher Weise der jeweilige Aufruf bearbeitet wurde. Bei regulärem Ablauf gilt: RET = Ø.

2.1 CREATE-Anweisung

Syntax:

```
CALL  CREATE  (DESK, FILE, RET)
```

wobei

```
DESK :  Datei-Deskriptor, beschreibt FILE
        mit
        DESK(1) : Satzlänge
            (2) : Anzahl der Sätze
          (3-5) : log. Gerätename
```

```
FILE :  Name der Datei
RET  :  Return-Code
```

Semantik:
Eine Datei mit Dateischutz und Eigentümerkennzeichen sowie den Werten des Dateideskriptors wird eingerichtet und dem DZS-Dateinamen FILE zugeordnet. Nachfolgende Dateiaufrufe benötigen FILE als Dateiidentifizierung. (ORG-Makro: $DAEI)

2.2 DELETE-Anweisung

Syntax:

```
CALL  DELETE  (FILE,RET)
```

wobei

```
FILE  :  Name der Datei
RET   :  Return-Code
```

Semantik:
Datei löschen, d.h. durch den Aufruf wird der physikalische Speicherplatz der Datei mit dem Namen FILE freigegeben.

Der Aufruf setzt eine im System eingerichtete und geschlossene Datei voraus, d.h. alle mit der Datei kommunizierenden Programme müssen die Datei freigegeben haben.
(ORG-Makro: $DALO)

2.3 OPEN-Anweisung

Syntax:

```
CALL  OPEN  (FILE, RET)
```

wobei
FILE : Name der Datei
RET : Return-Code

Semantik:
Vor den GET/PUT-Aufrufen auf eine Datei, muß diese durch eine OPEN-Anweisung eröffnet werden.
FILE gibt dabei den Namen der zu eröffnenden Datei an.
(ORG-Makro: $DAER)

2.4 CLOSE-Anweisung

Syntax:

```
CALL  CLOSE  (FILE, RET)
```

wobei
FILE : Name der Datei
RET : Return-Code

Semantik:
Schließen der Datei FILE, veränderte hauptspeicherpräsente Seiten werden vorher in die Datei FILE transferiert und alle Seiten der Datei aus dem Hauptspeicher eliminiert.
(ORG-Makro: $DASL)

2.5 GET-Anweisung

Syntax:

```
CALL  GET  (FILE, PAGE, BUFFER, RET)
```

wobei

```
FILE   :  Name der Datei
PAGE   :  Satznummer
BUFFER :  Feld im Anwenderprogramm
RET    :  Return-Code
```

Semantik:
Der Inhalt der Seite mit der Satznummer PAGE aus der Datei FILE wird in das Feld BUFFER eingetragen und die Satznummer PAGE um 1 inkrementiert.
(ORG-Makro: $STEIBI)

Hinweis:
Ein Plattenzugriff auf die Datei FILE findet durch den Aufruf nur dann statt, falls die Seite PAGE noch nicht hauptspeicher-präsent ist (siehe LRU-Algorithmus).

2.6 PUT-Anweisung

Syntax:

```
CALL  PUT  (FILE, PAGE, BUFFER, RET)
```

wobei

```
FILE   :  Name der Datei
PAGE   :  Satznummer
BUFFER :  Feld im Anwenderprogramm
RET    :  Return-Code
```

Semantik:
Den Inhalt des Feldes BUFFER in die Seite PAGE der Datei FILE schreiben und PAGE um 1 inkrementieren.
(ORG-Makro: $STAUBI)

Hinweis:
Die Seite PAGE wird durch den Aufruf noch nicht in die Datei transferiert (siehe LRU-Algorithmus).

3. Anwendung, Implementierung

Das DZS kann als eigenständiges Programm betrieben und von beliebig vielen Anwendern gleichzeitig angesprochen werden.

Es wird dabei wie ein Betriebsmittel im Sinne von Betriebssystemen an Anwender exklusiv vergeben.

Die Koordinierung der Kommunikation erfolgt mit Koordinierungszähler.

Bild 1 zeigt eine Mehrbenutzeranwendung.

DZS ohne AREA hat einen HSP-Bedarf von 3 k Worten. Die Größe von AREA ist bei der Programmierung anzugeben; dies ist bedeutsam, da auf Grund der Eigenschaften des LRU-Algorithmus durch eine Vergrößerung von AREA die Zahl der Peripherspeicherzugriffe verringert werden kann.

4. Beispiel: Benutzung einer temporären Datei.

Bild 2 zeigt Auszüge eines Fortran-Programms,das im folgenden kurz erläutert wird.
Mit dem CREATE-Aufruf wird eine Datei auf dem Peripherspeicher PLSK(0,0) mit 100 Sätzen und einer Satzlänge von 40 Worten eingerichtet.

Durch die PUT-Aufrufe wird die Datei sequentiell beschrieben und mit dem GET-Aufruf sequentiell gelesen. In dem Beispiel wird angenommen, daß der letzte Datensatz durch '-1' gekennzeichnet ist.

Der CLOSE-Aufruf bewirkt das schließen - und der DELETE-Aufruf das löschen der Datei.

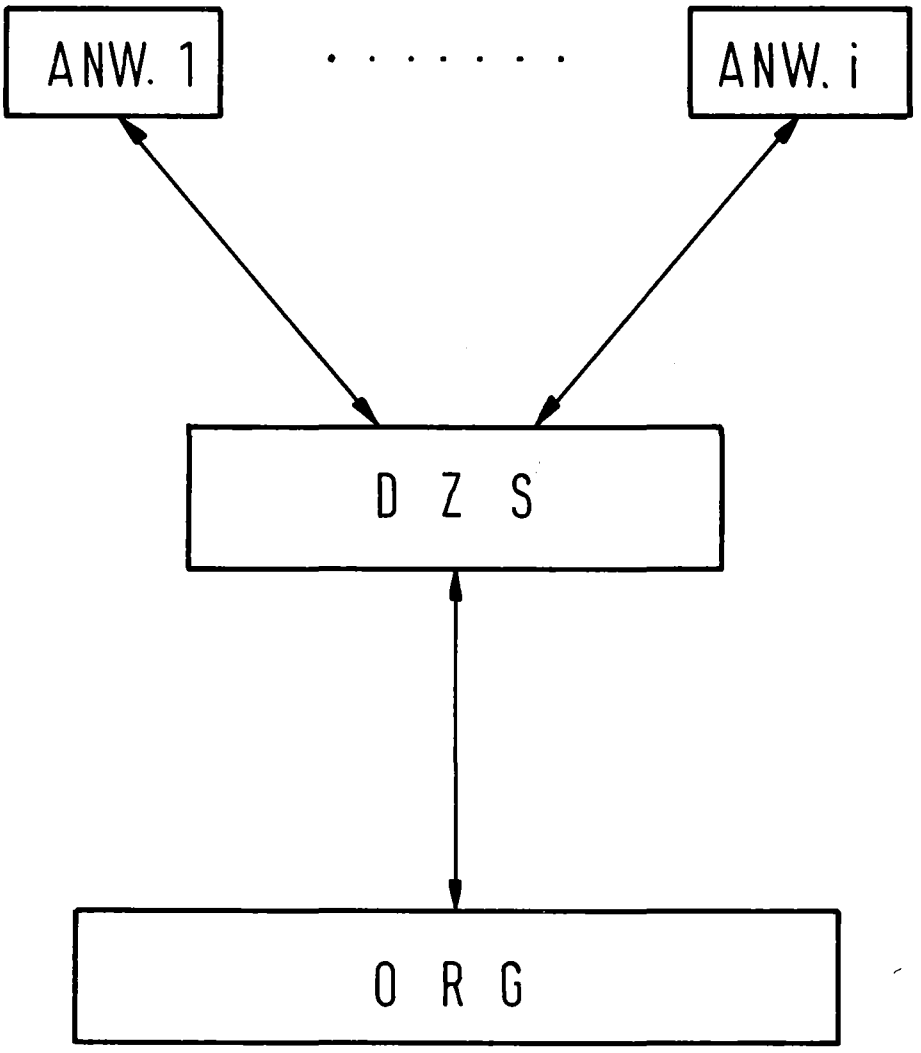


Bild 1.: Benutzung des DZS durch mehrere Anwender

```
C       DEFINITIONEN
C
        INTEGER*2 DESK(5),FILE,PAGE,RET,BUFFER(40)
        :
        DATA DESK /40,100,'PL','SK','00'/,  FILE /10/
C
C       PROGRAMM
        :
C       DATEI EINRICHTEN
C
        CALL CREATE (DESK,FILE,RET)
             IF (RET .NE. 0) GO TO 999
C
C       DATEI EROEFFNEN
C
        CALL OPEN (FILE,RET)
             IF (RET .NE. 0) GO TO 999
C
C       SAETZE SCHREIBEN
C
        BUFFER(I) = XYZ
C
        CALL PUT (FILE,PAGE,BUFFER,RET)
        :
C
C       SAETZE LESEN
C
   100      CALL GET (FILE,PAGE,BUFFER,RET)
C
            WRITE (7,500) BUFFER
            IF (BUFFER(1) .NE. -1) GO TO 100
C
C       DATEI SCHLIESSEN UND LOESCHEN
C
        CALL CLOSE (FILE,RET)
        CALL DELETE (FILE,RET)
        .
        .
        .
        .
```

Bild 2: Benutzung einer temporären Datei

L i t e r a t u r

/1/ Polster, F.-J.:
FADABS: Ein Datenbanksystem für den Siemens Prozeßrechner 330
In: Voges, U.:
Tagungsbericht der 9. Jahrestagung des Siemens Prozeßrechner-Anwenderkreises I,
Kernforschungszentrum Karlsruhe, 5.-7. April 1978
Kernforschungszentrum Karlsruhe, KfK 2642, S. 227-244

/2/ ORG 330 K
Beschreibung P71100-B0330-X-X-35
Siemens AG

Mikrocomputer im Rechen- und Steuerwerk einer DVA 301 zur Simulation einer DVA 305

U. Schwan*

1. Einleitung

Beim Arbeiten mit dem Siemens-Prozeßrechner DVA 301 macht sich der begrenzte Befehlsvorrat der DVA 301 gegenüber dem Prozeßrechner DVA 305 störend bemerkbar, da die beim Modell DVA 301 nicht verdrahteten Befehle durch Teile des Betriebssystems simuliert werden müssen. Daher dauert das Übersetzen und Rechnen in FORTRAN geschriebener Programme sehr lange, da hierzu der Befehlsvorrat der DVA 305 verwendet wird. Ein Zählprogramm für die Befehlssimulation zeigte, daß etwa 6/7 der gesamten Rechenzeit für die Simulation benötigt werden. Es sollte deshalb mit Hilfe von mikroprogrammierbaren, kaskadierbaren Mikroprozessorelementen das Rechen- und Steuerwerk der DVA 301 so umgebaut werden, daß es den Befehlsvorrat der DVA 305 verarbeiten kann. Neben dem Zeitgewinn bei der Befehlsausführung ergibt sich noch ein Gewinn an Kernspeicherplatz für Anwenderprogramme von etwa 700 Worten, da die zur Simulation benötigten Teile des Betriebssystems nicht mehr geladen werden müssen.

2. Struktur des Rechen- und Steuerwerkes der DVA 301

Die Zentraleinheit der DVA 301 (s. Bild 1) enthält folgende Funktionsgruppen:

(1) Arbeitspeicher
(2) Zentrales Element (Rechen- und Steuerwerk)
(3) Bedienungsfeld
(4) Stromversorgung

Im Arbeitsspeicher stehen die zu einem oder mehreren Programmen gehörigen Daten, das sind Binärmuster, die als Befehl, Dualzahl oder als Binärmuster interpretiert werden. Der Arbeitsspeicher ist ein Kernspeicher, er kann maximal 16 k (= 16384) Worte von 24 bit speichern. Ein- und Ausgabe erfolgen immer 24-bit-parallel, das Lese-Schreib-Register dient dabei zur Pufferung der Daten.

---

* cand.ing. U.Schwan ist studentische Hilfskraft am Lehrstuhl und Institut für Allgemeine Elektrotechnik und Automatisierung der RWTH Aachen, Direktor o.Professor Dr.-Ing.Hans Henning.
Der Mikrocomputer wurde im Rahmen einer Diplomarbeit gebaut.

Das Steuerwerk koordiniert den Datenverkehr des Arbeitsspeichers mit der Datennahtstelle, dem Rechenwerk und dem Bedienungsfeld. Es steuert den Ablauf eines Programms, sowie die Ausführung von Programmunterbrechungen.

Im Rechenwerk werden Transfer- und Verschiebebefehle, sowie logische und arithmetische Befehle ausgeführt. Es besteht im wesentlichen aus einem Verknüpfungswerk und dem Akkumulator. Beide verarbeiten 24-bit-parallel. Der Akkumulator stellt gleichzeitig die Verbindung zur PSF-Nahtstelle der DVA 301 dar.

Das Bedienungsfeld ermöglicht Eingriffe in den Ablauf der Zentraleinheit. Die Stromversorgung ist im selben Schrank untergebracht und versorgt die Zentraleinheit.

3. Daten und Aufbau des Mikrocomputers

Unter einem Mikroprozessor versteht man ein einzelnes integriertes Bauelement, das die Zentraleinheit für ein Prozessorsystem mit fester Datenbreite darstellt. Mikroprozessor-Elemente (slices) lassen sich aneinanderreihen, so daß ein Prozessor entsteht, der n·x bit verarbeiten kann, wobei n die Anzahl der Mikroprozessor-Elemente und x die Bitbreite eines Elementes ist.

Für die gestellte Aufgabe am besten geeignet erwies sich der Typ AM2901 der Firma Advanced Mikro Devices (AMD). Er ist mikroprogrammierbar, ist in bipolarer Technik aufgebaut, so daß sich eine genügende Schnelligkeit ergibt, und ist kompatibel zur TTL-Technik (Transistor-Transistor-Logik) der DVA 301. Sein innerer Aufbau (s. Bild 2) ermöglicht eine sehr flexible Anwendung, dadurch verringert sich der Aufwand an zusätzlichen Bauelementen. Der AM2901 verarbeitet 4 bit, es werden also 6 Einheiten benötigt, um das Rechen- und Steuerwerk der DVA 305 nachzubilden.

Aufbau des Mikrocomputers

Der Mikroprozessorbaustein AM2901 kann nur Mikrobefehle, das heißt grundsätzliche Befehle und Verknüpfungen ausführen. Wenn man nun ein Rechenwerk oder auch einen kompletten Rechner nachbilden (emulieren) will, so kann man die Maschinenbefehle meist nicht mit einem einzigen Mikrobefehl nachbilden, man benötigt dazu eine Folge von Mikrobefehlen, ein Mikroprogramm. Diese Mikroprogramme werden in Festwert-Speichern (ROM oder PROM) abgelegt. Da der AM2901 nur zur Datenverarbeitung ausgelegt ist, muß der Ablauf der Mikroprogramme durch eine Programmsteuerung kontrolliert werden. Jedes einzelne Mikroprogramm stellt nun einen Maschinenbefehl der DVA 305 dar, die Mikroprogramm-Steuerung muß daher

nicht nur den Ablauf der Mikroprogramme steuern, sondern auch die Zuordnung zwischen Maschinenbefehl der DVA 305 und dem entsprechenden Mikroprogramm. Dazu wird der Maschinenkode der DVA 305 in eine Startadresse des Mikroprogramms umgesetzt, das dann ablaufen soll.

Die Mikroprogramm-Steuerung besteht im wesentlichen aus drei Adressensteuerbausteinen AM2909, so können maximal 4 k Worte adressiert werden. Diese Bausteine ermöglichen eine flexible Programmierung wie z.B. die Verwendung von bedingten und unbedingten Sprüngen und maximal 4 verschachtelten Unterprogrammsprüngen. Ein Mikroprogrammwort ist 56 bit breit, davon dienen 22 bit der Programmsteuerung, 18 bit der Steuerung der 6 Mikroprozessoren, 3 bit sind frei für beliebige Anwendungen, die restlichen steuern die Peripherie der Mikroprozessoren, wobei 2 bit die Zentraleinheit der DVA 301 beeinflussen.

4. Einbau des Mikrocomputers in die DVA 301

Es gibt zwei sinnvolle Möglichkeiten des Einbaus des Mikrocomputers in die DVA 301. Er kann einmal alle Funktionen des Rechen- und Steuerwerkes übernehmen, zum anderen nur die Aufgaben des Rechenwerkes und die dazu benötigten Funktionen des Steuerwerkes. Im ersten Fall bietet sich die Einbaumöglichkeit an der Arbeitsspeicher-Nahtstelle (EXE-Nahtstelle) an, es wird dann nur noch der Kernspeicher der DVA 301 benutzt, das (alte) Steuerwerk stände fest auf Datenverkehr über EXE-Nahtstelle. Der Vorteil dieses Konzeptes ist es, daß kaum Umbauten in der Zentraleinheit der DVA 301 notwendig sind, lediglich die Datennahtstellen müßten geändert werden. Da jedoch zusätzlich der Wunsch bestand, das Bedienungsfeld der DVA 301 in seiner Funktion voll beizubehalten, wurde die zweite Möglichkeit gewählt.

Das Rechenwerk wird dabei vollständig ersetzt, das Steuerwerk der DVA 301 bleibt jedoch in seiner Funktion erhalten, es kann zusätzlich durch den Mikrocomputer beeinflußt werden, weil nur so Befehle ausführbar sind,die länger als ein Kernspeicherzyklus (1,6 µs) dauern (s. Bild 3). Dazu werden die Akkumulator-Platinen ausgebaut und stattdessen über Koppelplatinen der Mikrocomputer eingeschleift, die DVA 301 erhält also einen "intelligenten" Akkumulator. Der Mikrocomputer bildet dann mit Hilfe seiner internen Register den linken und den rechten Akkumulator, sowie das Exponentenregister für Gleitkommabefehle nach. Die Mikroprogrammsteuerung fragt das Steuerwerk der DVA 301 ab, um das Bitmuster im Lese-Schreib-Register dann als Befehl oder als Datenwort zu interpretieren. Gleichzeitig sorgt es durch Abfrage des internen Taktrasters der DVA 301 für die Synchronisierung zwischen beiden Geräten.

5. Technischer Aufbau des Mikrocomputers

Der Mikrocomputer ist auf einer doppelten und einer einfachen Europlatine in Mini-Wrap-Technik aufgebaut. Um die Mikroprogramme zu testen, wurde ein 32 mal 56 bit RAM (Lese-Schreib-Speicher) zusätzlich aufgebaut, in das über Tasten die Mikroprogrammworte eingegeben und über Leuchtdioden angezeigt werden können. So können die einzelnen Mikroprogramme unter Echtzeit getestet und geändert werden, bevor sie endgültig in PROM's abgelegt werden. Wahlweise ist der Betrieb im Einzelschritt oder mit dem Ur-Takt, also 5 MHz, der DVA 301 möglich. Außerdem ist es möglich, die Mikroprogramme in die RAM's zu laden oder den Inhalt der PROM's anzuzeigen. Die laufende Mikroprogrammadresse wird dabei, wie auch die zugehörige Startadresse, ebenfalls angezeigt. Das ermöglicht auch eine Fehlersuche im Betrieb. Die gesamte Stromaufnahme des Mikrocomputers und der Testhilfe beträgt etwa 9,5 bis 10,5 A bei 5 V Versorgungsspannung, daher muß eine externe Stromversorgung vorgesehen werden. Bei Versuchen im Laboraufbau wurde auch ermittelt, bis zu welcher Taktfrequenz die Mikroprogramme ablaufen können. Alle getesteten Programme liefen einwandfrei bis zu einer Taktfrequenz von 7 MHz, einige sogar bis zu einer Taktfrequenz von 10 MHz. Die kritischen Zeitbedingungen rühren nicht von den Mikroprozessorelementen, sondern von der Mikroprogrammsteuerung her. Das erklärt auch die unterschiedlichen Grenz-Taktfrequenzen, die von der Adressierungsart abhängig sind.

6. Ausführungszeiten der Mikroprogramme

Die Leistungsfähigkeit der AM2901-Bausteine läßt sich leicht ermessen, wenn man die Ausführungszeiten der umgebauten DVA 301 betrachtet. Dazu einige Beispiele:

Der Befehl VAR (Verschiebe Akkumulator arithmetisch rechts) benötigt folgende Ausführungszeiten:

- DVA 301 mit Simulation durch das Betriebssystem:
  für den linken Akkumulator nicht möglich
  für den rechten Akkumulator 178 bis 720 µs
- DVA 301 mit Mikrocomputer:
  für beide Akkumulatoren 3,2 bis 8 µs
- DVA 305 3,0 bis 37,5 µs
- DVA 306 1,8 bis 21 µs

Der Befehl DIV (Festpunkt-Division) benötigt:

- o DVA 301 mit Simulation:
  für beide Akkumulatoren 1333 µs
- o DVA 301 mit Mikroprozessor:
  für beide Akkumulatoren 6,4 µs
- o DVA 305 16,5 µs
- o DVA 306 9 bis 12,6 µs

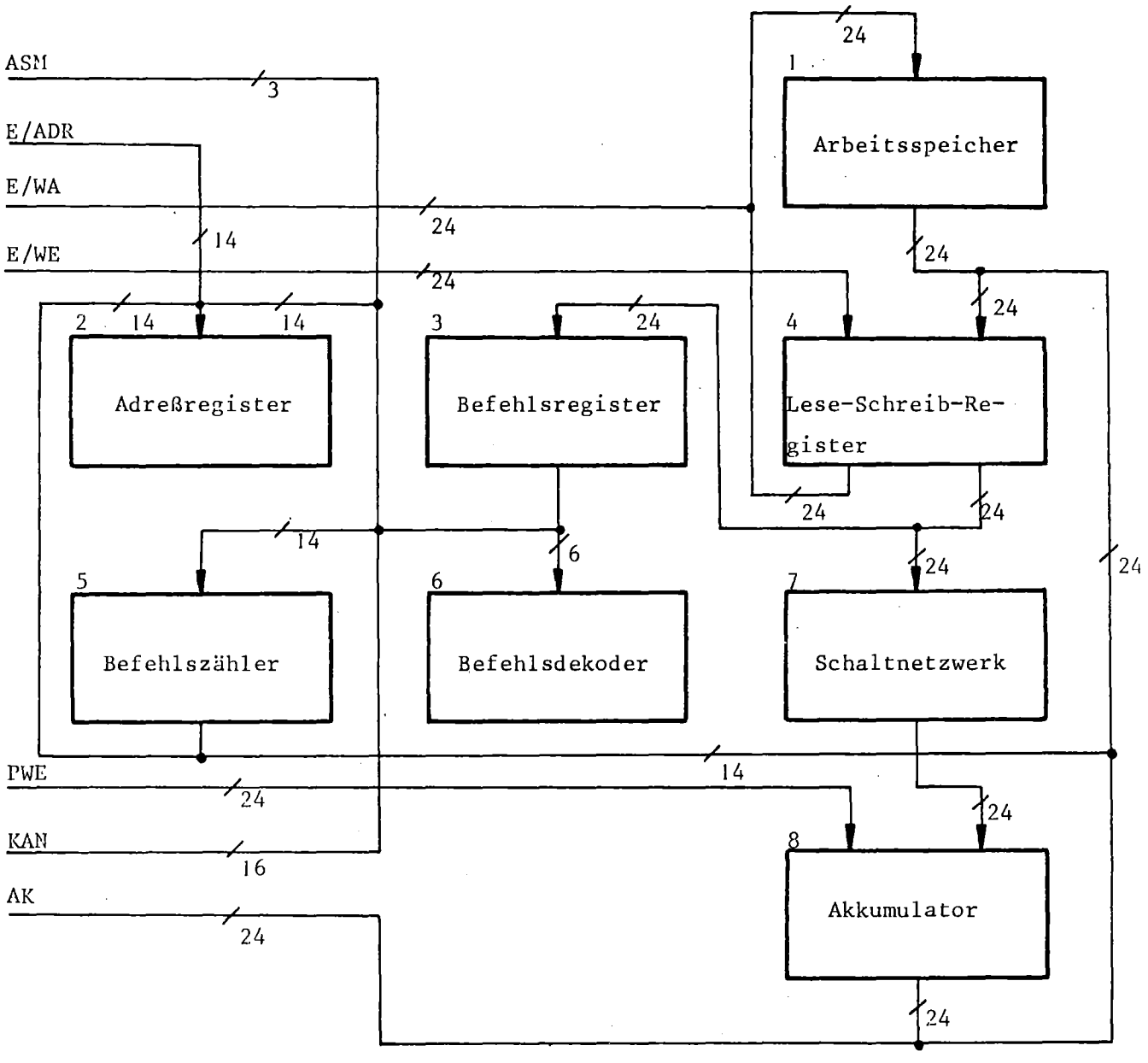
Der Mikrocomputer ist teilweise mehr als 200mal so schnell wie die DVA 301 bei Befehls-Simulation. Bei Befehlen, deren Ausführungszeiten nicht von der Zykluszeit des Kernspeichers abhängen, ist er wesentlich schneller als eine DVA 305 und sogar schneller als eine DVA 306.

Tabelle 1 enthält für den Befehlsvorrat der DVA 305 eine Gegenüberstellung der Ausführungszeiten bei einer DVA 301 mit Befehlssimulation, einer DVA 301 mit Mikrocomputer, einer DVA 305 und einer DVA 306.

7. Stand der Arbeit

Der Mikrocomputer und die Testhilfe sind fertiggestellt und in eine DVA 301 eingebaut. Zur Zeit werden die einzelnen Mikroprogramme erstellt und als einzelne Befehle in Echtzeit vom Bedienungsfeld aus getestet. Insbesondere muß untersucht werden, ob die Länge der Verbindungskabel, etwa 50 cm, zwischen Zentralem Element und Mikrocomputer unter ungünstigen Umständen Übertragungsfehler verursachen kann. Zusätzlich muß die Datenübertragung von und zur PSF-Nahtstelle getestet werden, da die Übertragung über den Mikrocomputer läuft.

Sobald der Befehlsvorrat der DVA 301 getestet ist, sollen die entsprechenden Mikroprogramme in die PROM's programmiert werden. Mit den speziellen Testprogrammen für die DVA 301 kann dann der gesamte Aufbau überprüft werden. Danach ist eine stufenweise Erweiterung des Befehlsvorrates bis zum Befehlsvorrat der DVA 305 vorgesehen, indem die einzelnen Mikroprogramme zunächst mit der Testhilfe ausgetestet und dann in die PROM's programmiert werden.



AK Datenausgang PSF-Nahtstelle

ASM Kanalauswahl EXE-Nahtstelle

E/ADR Arbeitsspeicheradresse von EXE-Nahtstelle

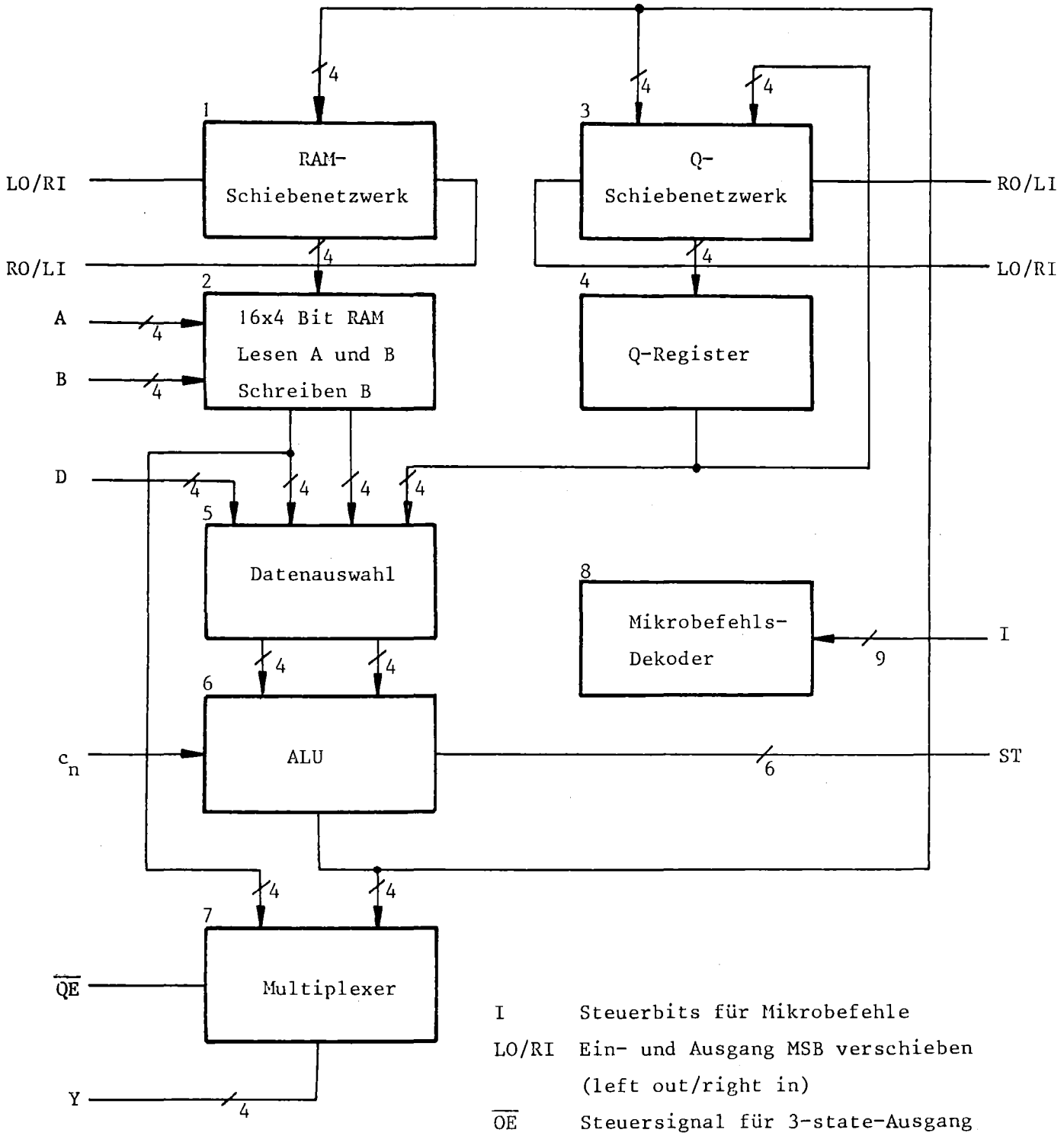
E/WA Wortausgang EXE-Nahtstelle

E/WE Worteingang EXE-Nahtstelle

KAN Parameter- und Kanalausgabe PSF-Nahtstelle

PWE Dateneingang PSF-Nahtstelle

Bild 1. Übersicht Zentrales Element der DVA 301



A Leseadresse

B Lese- und Schreibadresse

c_n Übertrag-Eingang

D Dateneingang

I Steuerbits für Mikrobefehle

LO/RI Ein- und Ausgang MSB verschieben (left out/right in)

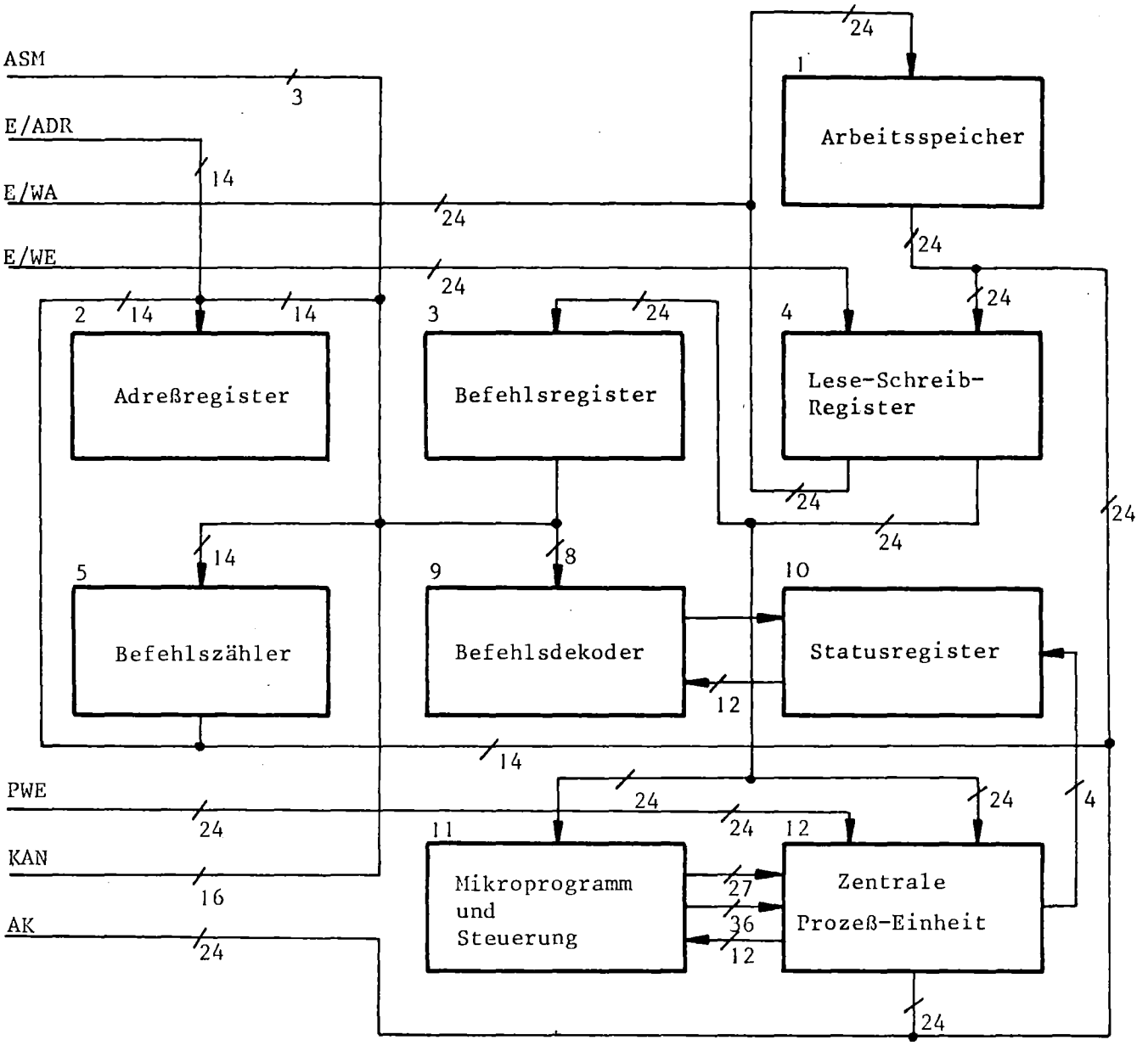
$\overline{OE}$ Steuersignal für 3-state-Ausgang

RO/LI Ein- und Ausgang LSB verschieben (right out/left in)

ST ALU-Zustand-Signale

Y Ausgang AM2901

Bild 2. AM2901 Mikroprozessor slice



AK Datenausgang PSF-Nahtstelle
ASM Kanalauswahl EXE-Nahtstelle
E/ADR Arbeitsspeicheradresse von ESE-Nahtstelle
E/WA Wortausgang EXE-Nahtstelle
E/WE Worteingang EXE-Nahtstelle
KAN Parameter- und Kanalausgabe PSF-Nahtstelle
PWE Dateneingang PSF-Nahtstelle

Bild 3. Umgebautes Zentrales Element der DVA 301

Tabelle 1. Befehlsausführungszeiten der Zentraleinheiten des Systems 300

Operationszeiten in µs

	DVA 301		DVA 301	DVA 305	DVA 306	
	LA	RA	und Mikrocomputer			
ADD	3,2	129	3,2	3,0	1,2	
ADT	107	142	4,8	4,5	2,4	
SUB	3,2	138	3,2	3,0	1,2	
MLT	978	-1138	6,4	16,5	8,4 - 12	!
DIV	1333		6,4	16,5	9 - 12,6	!
GAN	240 - 844		3,2 - 6,4[1)]	7,5 - 37,5	3 - 19,8	!
GSN	222 - 888		3,2 - 6,4[1)]	7,5 - 37,5	3 - 19,8	!
GMN	978	-1156	8 - 9,6[1)]	21 - 22,5	9,6 - 13,8	!
GDN	1333		8 - 9,6[1)]	19,5	10,2	
KPL	80	107	3,2	3,0	1,2	
ADA	116	150	1,6	1,5	0,6	
SBA	120	164	1,6	1,5	0,6	
LAP	111	124	1,6	1,5	0,6	
EHA	107	147	4,8	4,5	1,8	
ENA	107	151	4,8	4,5	1,8	
TEA	3,2	142	3,2	3,0	1,2	
UND	3,2	107	3,2	3,0	1,2	
ODR	3,2	107	3,2	3,0	1,2	
UGL	3,2	107	3,2	3,0	1,2	
ODL	102		3,2	3,0	1,2	

1) Ausführungszeiten sind geschätzte Werte

Fortsetzung Tabelle 1.

	DVA 301		DVA 301	DVA 305	DVA 306	
	LA	RA	mit Mikrocomputer			
VAL	-[2]	160 - 729	3,2 - 8	3,0 - 37,5	1,2 - 20,4	!
VAR	-[2]	178 - 720	3,2 - 8	3,0 - 37,5	1,8 - 21	
VLL	169 - 195[3]	178 - 213	3,2 - 8	3,0 - 37,5	1,2 - 20,4	!
VLR	164 - 195[3]	178 - 209	3,2 - 8	3,0 - 37,5	1,2 - 20,4	!
VDL	351 - 383	347 - 378	3,2 - 12,6	3,0 - 37,5	1,2 - 20,4	
VDR	364 - 396	360 - 391	3,2 - 12,6	3,0 - 37,5	1,2 - 20,4	
VSE	156 - 333	169 - 338	3,2 - 8	4,5 - 18,0	1,8 - 9	
TEP	3,2	93	3,2	3,0	1,2	
TEL	3,2	102	3,2	3,0	1,2	
TEM	3,2	98	3,2	3,0	1,2	
TAS	3,2	107	3,2	3,0	1,2	
TEX	93		3,2	3,0	1,2	
TAX	102		3,2	3,0	1,2	
SPR	1,6	NNN	1,6	1,5	0,6	
SGN	1,6	111	1,6	1,5	0,6	
SUN	1,6	111	1,6	1,5	0,6	
SAP	1,6	111	1,6	1,5	0,6	
SAM	1,6	111	1,6	1,5	0,6	
SUL	-[2]	98	1,6	1,5	0,6	!
SKU	-[2]	102	1,6	1,5	0,6	!
UNT	3,2	NNN	3,2	3,0	1,2	

[2] Diese Befehle können von der DVA 301 nicht ausgeführt werden

[3] Verschiebezahl v ≠ 1

Kurven- und Prozeßsichtgerät für eine DVA 301

W. Kristen*

1. Einleitung

In zunehmendem Maße werden Datensichtgeräte als Kommunikationsmittel zwischen Rechner und Mensch eingesetzt. Ihre Vorteile gegenüber den bisher benutzten mechanischen Geräten liegen in der geräuschlosen Arbeitsweise und ihrer Schnelligkeit. Alphanumerische Sichtgeräte werden anstelle eines Blattschreibers dort angeschlossen, wo kein Papier als Beleg für den Datenverkehr mit dem Rechner benötigt wird. Kurvensichtgeräte werden in Anlagenwarten als Ersatz für die mechanischen Meßwertschreiber benutzt. Prozeßsichtgeräte ersetzen zunehmend in Warten die bisher in Mosaiktechnik ausgeführten Anlagenübersichten, die meist eine große Wandfläche belegen. Graphische Sichtgeräte, die wie X-Y-Schreiber die Darstellung jeder beliebigen mathematischen Funktion ermöglichen, werden zur graphischen Auswertung wissenschaftlicher Experimente verwendet.

Da die zur Anschaffung von Sichtgeräten benötigten Geldmittel nicht zur Verfügung standen, es aber wünschenswert erschien, solche Geräte an die Rechenanlage des Instituts anzuschließen, um sie bei der Ausbildung von Studenten einsetzen zu können, wurde im Rahmen einer Diplomarbeit ein Kurvensichtgerät entwickelt und gebaut. Es ermöglicht die Ausgabe von alphanumerischen Zeichen und die Wiedergabe von maximal vier Meßwertverläufen in Kurvenform auf einem handelsüblichen Farbfernsehgerät. Auf der SAK-Tagung 1977 in Dortmund hat Herr John über das Kurvensichtgerät berichtet.

Inzwischen wurde dieses Sichtgerät im Rahmen einer weiteren Diplomarbeit zum Prozeßsichtgerät weiterentwickelt. Über frei generierbare Sonderzeichen ist die Darstellung von Anlagenübersichten und Schaltplänen möglich. Der Bildaufbau erfolgt mit Hilfe eines Lichtgriffels.

---

* Dipl.-Ing.W.Kristen ist wissenschaftlicher Mitarbeiter am Lehrstuhl und Institut für Allgemeine Elektrotechnik und Automatisierung der RWTH Aachen, Professor Dr.-Ing.Hans Henning.

Das Sichtgerät wurde im Rahmen ihrer Diplomarbeiten von Herrn F.Walbrodt und Herrn W.Merten gebaut.

2. Technische Daten des Kurvensichtgerätes

Da bei der Entwicklung des Prozeßsichtgerätes möglichst viele Funktionen und Baugruppen des Kurvensichtgerätes beibehalten werden sollten, werden die Darstellungsmöglichkeiten und technischen Daten des Kurvensichtgerätes kurz aufgeführt und erläutert:

- Zeichenerzeugung: Die Zeichenerzeugung erfolgt nach dem Fernsehprinzip in einem 5 x 7 Raster mit einem zusätzlichen Rasterpunkt Abstand zum nächsten Zeichen derselben Reihe und drei Rasterpunkten Abstand zum nächsten Zeichen der nächsten Reihe.
- Zeichenvorrat: Der Zeichenvorrat umfaßt die 64 Zeichen des ASCII-Codes.
- Zeichenwiedergabe: Es können alphanumerische Zeichen in 26 Reihen mit 64 Zeichen pro Reihe auf einem Farbmonitor oder einem handelsüblichen Farbfernsehgerät abgebildet werden. Jedes Zeichen kann in einer von 7 Farben erscheinen. Die Farbe kann von Zeichenplatz zu Zeichenplatz wechseln. Die Zeichenausgabe kann ausgehend von der linken oberen Bildecke oder ab einer frei wählbaren Stelle erfolgen.
- Roll Up: Ist der gesamte Bildschirm mit Zeichen vollgeschrieben, so wird vor Ausgabe der nächsten Zeichenreihe der gesamte Bildinhalt um eine Reihe nach oben geschoben. Der Inhalt der 1. Zeichenreihe geht verloren, die neuen Zeichen erscheinen in der zuvor gelöschten 26. Zeichenreihe.
- Blinker: Ein blinkender Strich kennzeichnet den Zeichenplatz, an dem das nächste Zeichen erscheinen wird.Dieser Blinker kann durch Steuerbefehle verschoben oder durch Angabe der Reihennummer und der Platznummer an eine bestimmte Stelle gesetzt werden (frei positionierbare Zeichenausgabe).
- Kurvenfeld: In das Zeichenfeld kann ein Kurvenfeld eingeblendet werden, das in der Waagerechten (Amplituden-Achse) 256 Rasterpunkte und in der Senkrechten (Zeitachse) 200 Rasterpunkte umfaßt.
- Raster: Das Kurvenfeld kann durch ein Raster unterteilt werden. Die senkrechten Rasterlinien (Amplitudenraster) können einen Abstand von 32 oder 50 Rasterpunkten haben, die waagerechten Rasterlinien (Zeitraster) einen Abstand von 20 oder 50 Rasterpunkten. Die Rasterlinien können in einer von 7 Farben erscheinen oder ganz unterdrückt werden. Außerdem besteht die Möglichkeit, nur die äußere Umrandung des Kurvenfeldes und die Kreuzungspunkte der Rasterlinien einzublenden. Bei der Darstellung bipolarer Meßwerte wird die Mittellinie des Amplitudenrasters durch doppelt breite Darstellung hervorgehoben.

- Meßwertdarstellung: Im Kurvenfeld können maximal 4 Meßwertverläufe als Funktion der Zeit dargestellt werden. Jede Meßwertkurve kann in einer von 7 Farben erscheinen. Es können unipolare Meßwerte im Zahlenbereich von 0 bis 255 oder bipolare Meßwerte im Zahlenbereich von -128 bis +127 dargestellt werden. Der aktuelle Meßwert erscheint am unteren Rand des Kurvenfeldes. Durch jeden neuen Meßwert werden die alten Meßwerte um einen Rasterpunkt nach oben verschoben. Sind bereits 200 Meßwerte dargestellt, so wandert der älteste Meßwert aus dem Bild.

  Ergänzend zur punktweisen Darstellung können die einzelnen Meßwerte durch Linien verbunden werden oder die Fläche zwischen der Nullinie und den Meßwerten hellgetastet werden. Zur Kennzeichnung eines besonderen Zustandes (Alarm) kann jede Kurve unabhängig von den anderen blinkend dargestellt werden. Jede Kurve kann einzeln in das Kurvenfeld ein- oder ausgeblendet werden.
- Zeichenpriorität: Zeichen im Kurvenfeld unterdrücken die Kurvendarstellung.
- Rechnerschnittstelle: Der Datenverkehr erfolgt über eine 9bit breite parallele Schnittstelle (ZAP-Schnittstelle der DVA 301) nur in der Richtung vom Rechner zum Kurvensichtgerät. Die maximale Datenübertragungsrate liegt bei ca. 1000 Zeichen pro Sekunde.

3. Technische Daten des Prozeß-Sichtgerätes

Die Weiterentwicklung zum Prozeß-Sichtgerät machte einige Änderungen und Ergänzungen der Daten des Kurvensichtgerätes erforderlich. Sie sind nachstehend aufgeführt und erläutert.

- Zeichenabstand: Der Abstand zwischen zwei alphanumerischen Zeichen im 5 x 7-Raster wurde zum nächsten Zeichen derselben Reihe auf zwei Rasterpunkte erhöht und der Abstand zum nächsten Zeichen der nächsten Reihe auf zwei Rasterpunkte verkleinert.
- Reihenzahl: Wegen dem geringeren Abstand zwischen den Zeichenreihen wurde die Zahl der Zeichenreihen von 26 auf 28 erhöht.
- Darstellungsart: Zusätzlich zur frei wählbaren Farbe kann jedes Zeichen normal oder invertiert (z.B. als rotes Zeichen vor schwarzem Hintergrund oder als schwarzes Zeichen vor rotem Hintergrund) und nicht blinkend oder blinkend dargestellt werden.

- Zeichenvorrat: Der darstellbare Zeichenvorrat der 64 ASCII-Zeichen wurde um drei Gruppen von je 64 Sonderzeichen im 7 x 9-Raster erweitert. Die für den Haupteinsatz des Prozeß-Sichtgerätes benötigte Gruppe von Sonderzeichen kann in einem programmierbaren Nur-Lese-Speicher (PROM) abgespeichert werden. Die beiden anderen Gruppen stehen in Schreib-Lese-Speichern (RAM), deren Inhalt vom Rechner aus verändert werden kann. Damit steht dem Anwender ein beliebig großer Vorrat von Sonderzeichen zur Prozeß- oder Schaltplan-Darstellung zur Verfügung.
- Bildaufbau: Der Bildaufbau bei der Darstellung von Prozeß-Abläufen oder Schaltplänen erfolgt mit einem Lichtgriffel. Hierzu werden in den Reihen 1 bis 24 in der Mitte jedes Zeichenplatzes ein heller Punkt, in der 25. Zeichenreihe der alphanumerische Zeichenvorrat und in den Reihen 26 bis 28 die Sonderzeichengruppen eingeblendet. Durch Berühren eines Zeichens mit dem Lichtgriffel und anschließendem Berühren des Punktes auf einem Zeichenplatz wird das gewünschte Zeichen an diesen Platz übertragen. Ein Zeichen wird solange in die berührten Zeichenplätze übertragen, bis mit dem Lichtgriffel ein neues Zeichen berührt wird.
- Bildspeicherung: Ein mit dem Lichtgriffel in den Zeichenreihen 1 bis 24 aufgebautes Bild ist aus dem Bildspeicher des Prozeß-Sichtgerätes in den Hauptspeicher des Rechners oder auf einen Externspeicher zur Bildarchivierung auslesbar. Bei Bedarf kann das gesamte Bild dann wieder an das Prozeß-Sichtgerät übergeben werden.
- Meßwertdarstellung: Bei der Meßwertwiedergabe erscheint der aktuelle Meßwert jetzt am oberen Bildrand (bisher am unteren). Diese Art der Meßwertdarstellung entspricht einer Drehung des Koordinatensystems von 90^{o} im Uhrzeigersinn und der bei mechanischen Kurvenschreibern üblichen Darstellungsart.
- Rechnerschnittstelle: Um eine Bildeingabe in den Rechner zu ermöglichen, wurde der ZAP-Schnittstelle zur Datenübertragung vom Rechner zum Sichtgerät eine 9bit breite parallele Schnittstelle zur Datenübertragung vom Sichtgerät zum Rechner hinzugefügt (ZEP-Schnittstelle bei der DVA 301).

4. Aufbau des Gerätes

Das Prozeßsichtgerät arbeitet nach dem in Bild 1 gezeigten Schema. Die vom Rechner kommenden Daten werden im Daten-Steuerwerk in Befehle und Informationen umgeschlüsselt und auf getrennten Datenwegen (Englisch: bus) weitergeleitet. Befehle steuern die Funktion des Gerätes und die Weitergabe und Speicherung der Daten des Informations-Busses.

Im Kurventeil werden das Kurvenraster erzeugt und die Meßwerte gespeichert. Eine Rückübermittlung der im Kurventeil gespeicherten Meßwerte an den Rechner ist nicht möglich.

Im Bildteil werden die Erzeugung der alphanumerischen und der Sonderzeichen gesteuert und die Zeichen eines Bildes gespeichert. Zwei Gruppen der Sonderzeichen stehen in Schreib-Lese-Speichern (RAM), die eine Zykluszeit von unter 100 ns haben. Da diese Schaltkreise sehr teuer sind, wurde eine Lösung gesucht, die eine platzsparende, verschachtelte Speicherung des Zeichenvorrates in vier 256 x 4 bit-RAM's ermöglicht.

Die im Bildspeicher stehende Information kann an den Rechner übertragen werden. Da die Schnittstelle vom Sichtgerät zum Rechner nur 8bit breit, jedes Zeichen aber mit 13bit codiert ist, mußte das Auslesen des Bildspeichers in den Rechner so gesteuert werden, daß zuerst der Zeichencode (8bit) und anschliessend Zeichenfarbe und Darstellungsart (5bit) übertragen werden.

Die aus Bild- und Kurventeil kommenden Signale werden zu einem normgerechten Bildsignal für ein Farbfernsehgerät zusammengesetzt.

Über einen Lichtgriffel und ein zugehöriges Steuerteil ist ein rechnerunabhängiger Aufbau eines Bildes möglich. Mit dem Lichtgriffel kann aber nur der Code des Sonderzeichens erfaßt werden. Deshalb ist der Lichtgriffel über ein zusätzliches Bedienpult an das Prozeß-Sichtgerät angeschlossen. An diesem Pult können Farbe und Darstellungsart für jedes Zeichen über Schalter vorgewählt werden. Außerdem können über einen Schalter die Markierung der Zeichenplätze und der Zeichenvorrat eingeblendet werden.

5. Programmausrüstung (Software)

Für das Kurvensichtgerät wurde, in Anlehnung an das Programm MAKUSAO der Firma Siemens, ein Paket von Makroaufrufen geschrieben, die eine Vorwahl der gewünschten Funktion und die Bild- und Kurvenausgabe ermöglichen. Diese Makroaufrufe gelten auch für das Prozeß-Sichtgerät. Für das Prozeß-Sichtgerät wurden

aus Zeitgründen bisher erst zwei kleine Programme geschrieben, die das Laden der Sonderzeichen von Lochkarten und eine Bildarchivierung auf Lochstreifen ermöglichen.

Laden von Sonderzeichen

Um den Sonderzeichenvorrat platzsparend in RAM's abspeichern zu können, wurde eine verschachtelte Speicherung in vier 256 x 4bit-RAM's aufgebaut. Die Dateneingabe eines Sonderzeichens in diese verschachtelten Speicher ist aber einem Anwender nicht zumutbar. Deshalb wurde folgender Weg beschritten: Der Anwender bestimmt, welche Bildpunkte hellgetastet werden sollen. Diese setzt er gleich Null, die übrigen gleich Eins. Diese Bitmuster ergänzt er links um eine führende Eins. Anschließend stanzt er die Bitmuster jeder Zeile, immer links beginnend, ohne Zwischenraum hintereinander in eine Lochkarte. Der Zeichenvorrat einer Zeichengruppe ist somit in 64 Lochkarten, die in den ersten 72 Spalten gelocht sind, enthalten. Ein Programm übernimmt dann das Einlesen der Lochkarten, das Umsetzen in den Binärcode und das verschachtelte Laden des Zeichenspeichers.

Bildarchivierung

Ein mit dem Lichtgriffel im rechnerunabhängigen Betrieb aufgebautes Bild kann aus dem Bildspeicher des Sichtgerätes an den Rechner übergeben werden. Hierbei wird erst der Zeichencode in dem Sichtgeräte-internern 8bit-Code übergeben und dann Farbe und Darstellungsart. Ein Zeichen belegt somit 16bit im Hauptspeicher des Rechners. Da der Prozeßrechner des Institutes aber eine Wortlänge von 24bit hat, wird, um Speicherplatz zu sparen, der Bildspeicher byteweise so ausgelesen,daß drei Zeichen zwei Speicherplätze des Rechners belegen. Anschließend kann der Bildinhalt zur Archivierung auf Lochstreifen gestanzt werden.

Beim Einlesen eines Bildes von einem Lochstreifen muß per Programm die eng gepackte Speicherung wieder aufgehoben, die Information der Darstellungsart in Steuerbefehle für Farbe, Blinken und Invertierung und der interne Zeichencode in den ASCII-Code umgeschlüsselt werden.

Bild-Bibliothek

Es ist geplant, ein Programm zur Archivierung von Sonderzeichenvorräten und kompletten Bildern auf einem Plattenspeicher zu schreiben, damit ein schneller Bildwechsel möglich wird.

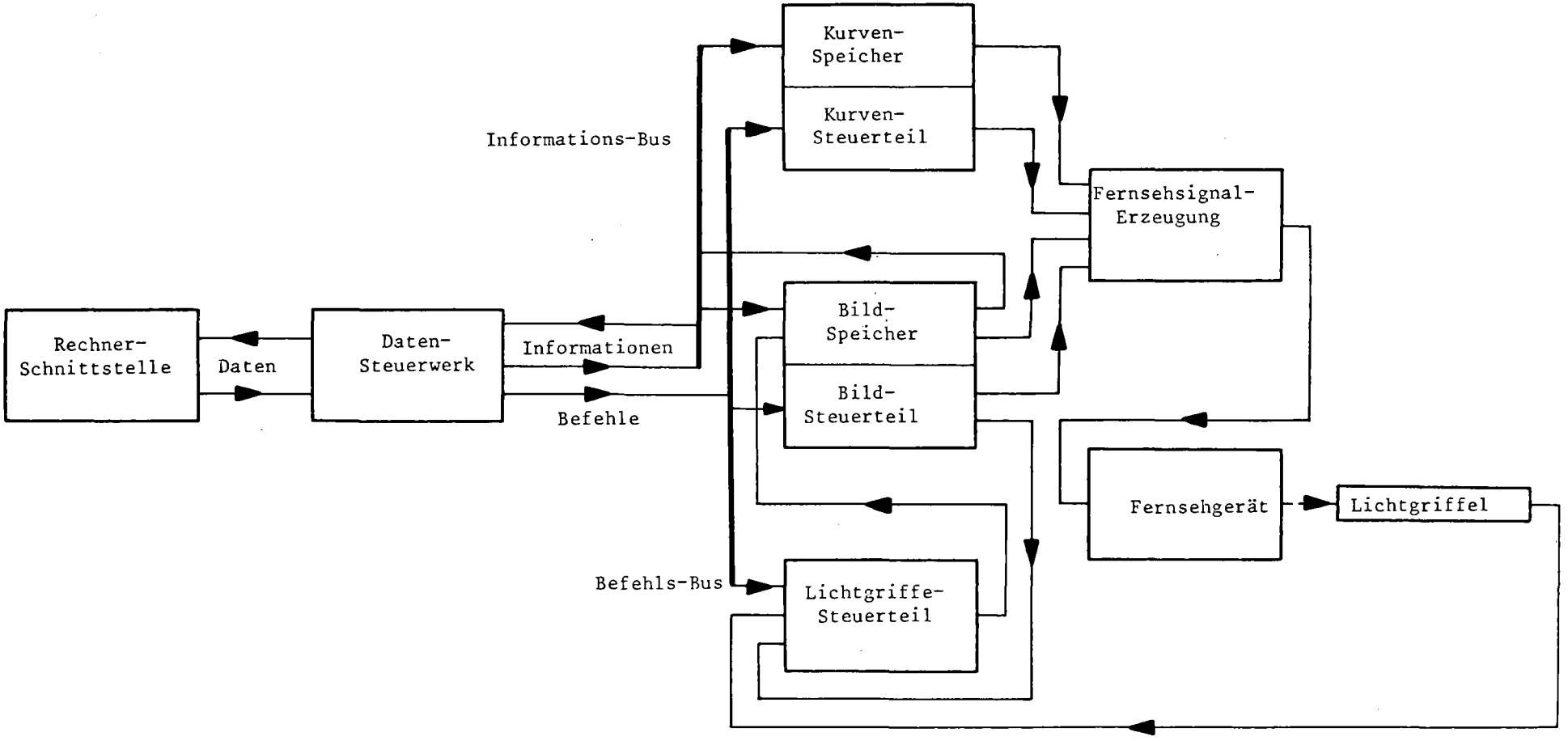


Bild 1. Funktionsschema des Prozeß-Sichtgerätes

SPEICHERERWEITERUNG FÜR DIE SIEMENS 306

H. Huppertz, P. Reinhart, F. Rongen, W. Tenten

Betrachtet man die entwicklungsmäßig abgeschlossene Typenreihe der 24 Bit Rechner des Siemens Systems 300, so muß man feststellen, daß das Modell 306 leistungsmäßig am oberen Ende dieser Reihe steht. Die 306 ist nicht nur im Prozeßlenkungsbereich zu finden, sondern es sind auch einige Maschinen im Instituts-bereich eingesetzt. In beiden Fällen fallen dem Rechner sehr unterschiedliche Aufgaben zu. Sie reichen z.B. im Institutsbereich von der Speicherung großer Datenmengen bis zur Auswertung von Experimentdaten mit zum Teil sehr umfangreichen Rechenprogrammen. Trotzdem ist selbst eine voll ausgebaute 306 für den einen oder anderen Anwender nicht ausreichend, weil der Vollausbau schon bei 64K Kernspeicher erreicht ist und auch die als Externspeicher verwendeten Wechselplatten verhältnismäßig klein sind. Als Alternative stehen aber nur die 16 Bit Rechner mit höherer Kernspeicherausbaufähigkeit und wesentlich größeren Externspeichern zur Verfügung. Bei sehr viel anwendungsbezogener Software wäre eine Umstellung auf einen 16 Bit Rechner mit einem unzumutbaren Kostenaufwand verbunden. Zum anderen stellt sich die Frage der Hardwareinstrumentierung an den Schnittstellen der Prozeßelemente, die, zum Teil selbst gebaut, zum Teil vom Hersteller bezogen, dann ebenfalls erneuert werden müßten.

Die Konfiguration der im Zentrallabor für Elektronik der Kernforschungsanlage Jülich installierten Anlage ist in Abb. 1 dargestellt. Sie hat eine Arbeitsspeicherkapazität von 48K. Über die langsame Standardnahtstelle sind die Ein- Ausgabeeinheiten, wie Bedienungsblattschreiber, Lochkartenleser, -stanzer, Lochstreifenleser, -stanzer, Schnelldrucker und ein selbst entwickeltes Rechnerkopplungsinterface, über das mehrere PDP8 Rechner angekoppelt sind, angeschlossen. Der Schnellkanal, der es erlaubt, max. 5 externe Elemente anzuschließen, ist voll ausgebaut, und zwar mit einer Kopplung zur IBM 370/168, einer Zwillingsbandeinheit, zwei Plattenspeichern und einer Kopplung zum CAMAC-Datenweg, über den die 306 mit einer 320 verbunden ist.

Am CAMAC-Datenweg sind über serielle Schnittstellen mehrere Sichtgeräte sowie Vielkanalanalysatoren angeschlossen.

Neben den Real-Time-Anforderungen durch angeschlossene Experimentrechner und sonstige Experimentiergeräte stehen die in einem Time-Sharing-Dialog-Verkehr betriebenen graphischen Sichtgeräte gegenüber. Neben dem On-line-Betrieb wird die 306 in immer stärkerem Maße als Betriebsrechner für nicht zeitkritische Aufgaben eingesetzt. Bei dem jetzigen geringen Kernspeicherausbau steht in der Maschine aber nur ein großer Laufbereich von 16K zur Verfügung, der sowohl von den Betriebsprogrammen wie auch sehr stark von den On-line Programmen genutzt wird. Durch das häufige Umspeichern der Programme auf die Platte beim Laufbereichswechsel entstehen für den On-line Betrieb sehr lange Reaktionszeiten.
Abhilfe kann hier nur geschaffen werden

1. durch schnellere Systemspeicher,
2. durch Schaffen mehrerer 16K Laufbereiche, wobei der Ausbau bis 64K nur eine unbefriedigende Lösung brächte.

Für den Anschluß von Kernspeichern an das Rechensteuerwerk dienen 4 Nahtstellen. Der maximale Speicherausbau der 306 von 64K setzt sich zusammen aus 4 Modulen à16K, die an diese Nahtstellen angeschlossen sind. Jeder 16K Block ist eine in sich abgeschlossene Einheit, d.h. nachdem das Steuerwerk einen bestimmten Block ausgewählt hat, wird im Speichermodul ein Zeitraster erzeugt, wonach die für Kernspeicher typische Lese- und Wiedereinschreibphase abläuft. Als Modus kann der Speicher folgende Operationen ausführen

1. Lesen
2. Schreiben
3. Lesen/Löschen

Adressiert wird der Speicher durch das 17 Bit lange Adreßregister im Steuerwerk. Der vom Hersteller geplante Ausbau auf maximal 128K wird so durchgeführt, daß an Stelle der 16K Module solche mit doppelter Kapazität an die 4 vorhandenen Nahtstellen angeschlossen werden. Dabei können 16K und 32K Blöcke nicht gemischt werden.Weil ein Kernspeicher zu Siemens-Preisen indiskutabel ist und auf dem OEM-Markt 25 Bit Kernspeicher so gut wie nicht erhältlich sind, haben wir uns entschlossen, einen preislich günstigeren Halbleiterspeicher zu entwickeln, bei dem weder Wortlänge noch Zugriffszeit große Probleme aufwerfen.
Der Halbleiterspeicher wurde schnittstellenkompatibel zu den von Siemens eingesetzten 16k Kernspeicherblöcken aufgebaut. Hierdurch gewinnt man den Vorteil, daß jedes der 16K Kernspeichermodule durch einen Halbleiterspeicher ersetzt werden kann. Beim Ausbau größer 64K müssen im Steuerwerk geringfügige Modifikationen vorgenommen werden, damit einmal an eine Nahtstelle mehr als 16K angeschlossen werden können und zum anderen keine gravierenden Eingriffe ins Betriebssystem erforderlich sind. Die Erweiterung kann beliebig an jeder der 4 Nahtstellen erfolgen (Abb.2). Ist eine Maschine bereits mit 64K Kernspeicher ausgerüstet, so wird der 16K Kernspeicherblock an Nahtstelle 4 stillgelegt und durch die Halbleiterspeichererweiterung ersetzt. Obwohl es möglich ist, die 306 nur mit Halbleiterspeichern auszurüsten, besteht der typische Ausbau wohl aus Kern- und Halbleiterspeichern.

Da der gesamte Zyklus zwischen Steuerwerk und den Speichermodulen in einem starren Zeitraster ablaufen muß, konnten nur statische Halbleiterspeicher verwendet werden. Die zur Zeit auf dem Markt erhältlichen Bausteine, die in ihren technischen Daten den Anforderungen entsprechen, sind 4K statische MOS-RAM's mit einer Zugriffszeit von 200 nsec, die bereits in einem Halbleiterspeichermodul für die 16 Bit Rechner eingesetzt wurden. (1)

Aufgebaut wurde der Halbleiterspeicher auf Doppeleuropakarten, die über 96-polige Federleisten auf einer als Bus verdrahteten Standardrückwandplatine gesteckt werden. Man kann den Halbleiter-

speicher funktionell aufteilen in:

1. Steuerteil,
2. Speicherteil.

Der Steuerteil generiert in Abhängigkeit vom angelegten Modus die für den Speicher benötigten Lese- Schreibsignale. Zur Anpassung der TTL-kompatiblen Speicherbausteine an die ECL-Rechnerlogik werden entsprechende Umsetzer eingesetzt. Außerdem ist für die Erweiterung über 64K eine vollständige Adreßdekodierung erforderlich.
Der Speicherteil beinhaltet nur die eigentlichen 4K-MOS-RAM's. Für eine 16K Einheit benötigt man 1oo Bausteine, die auf 2 Karten aufgebaut sind. Weil die 3o6 nur um 16K Blöcke erweitert werden kann, wurden die beiden Platinen zu einer Einheit zusammengefügt und als Sandwich-Baugruppe nur mit einer Federleiste versehen. Über einen Schiebeschalter kann ein Adreßbereich zwischen 0-128K eingestellt werden. Da die Standardrückwandplatine als Bus ausgelegt ist, kann durch Hinzufügen von 16K Einheiten die Maschine beliebig bis 128K ausgebaut werden (Abb. 3).

Um dem Engpaß auf dem Externspeicher-Sektor zu begegnen, wird zur Zeit untersucht, ob man einen dem Stand der Technik entsprechenden Magnetplattenspeicher anschließen kann. Die zur Zeit existierenden Plattenspeicher an der 306 haben eine Kapazität von $1{,}6 \times 10^6$ Worte pro Laufwerk. Für jedes Laufwerk wird eine Steuerung und auch eine Schnellkanal-Nahtstelle benötigt. Stand der Technik sind heute Plattenspeicher von 50×10^6 Worte pro Laufwerk, wobei man 4-8 Laufwerke pro Steuerung anschliessen kann. Bei einer Erweiterung der Plattenkapazität auf der Basis der 306 Laufwerke muß die Steuerung so umgebaut werden, daß man mehrere Laufwerke pro Steuerung anschliessen kann. Das macht aber nur den Faktor 4-8 aus. Dem gegenüber erreicht man mit dem Anschluß einer modernen Platte eine Kapazitätserweiterung um den Faktor 32. Bei universeller Auslegung der Steuerung erreicht man hier

zusätzlich nochmal den Faktor 4-8. Die Steuerung wird systemkonform an die Schnittstelle des Prozeßelementes P3KS angepaßt. Zur Adressierung des Plattenspeichers wird neben dem vorhandenen Programmkanal zusätzlich das Maskenregister benutzt. Der erforderliche Eingriff ins Betriebssystem ist nur geringfügig, hat aber keinen Einfluß auf die Benutzersoftware. Für den Anwender sieht der Anschluß so aus, als ständen ihm mehrere PSP bzw. PSK Plattenspeicher zur Verfügung.

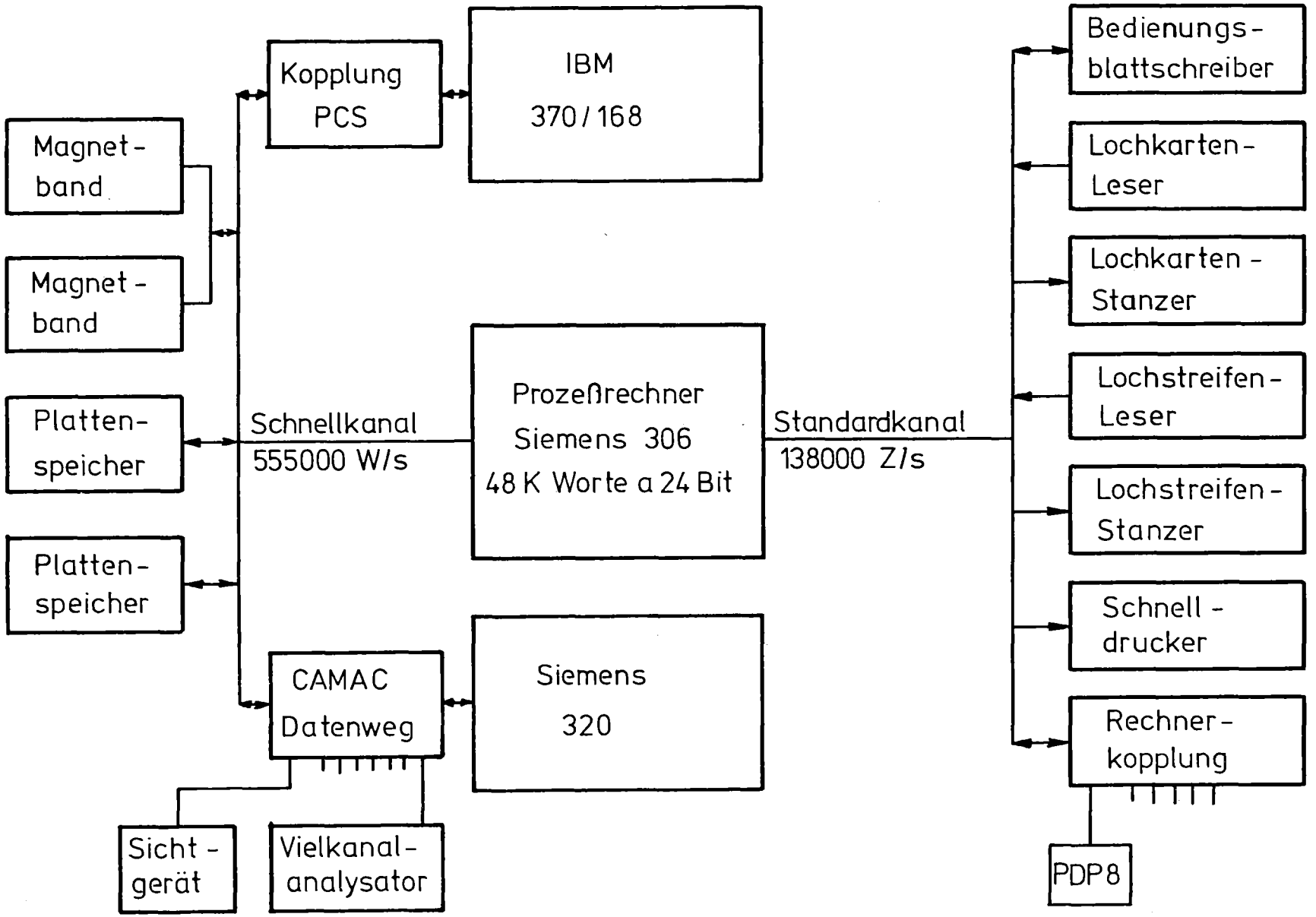


Abb.1 Anlagenkonfiguration der Siemens 306

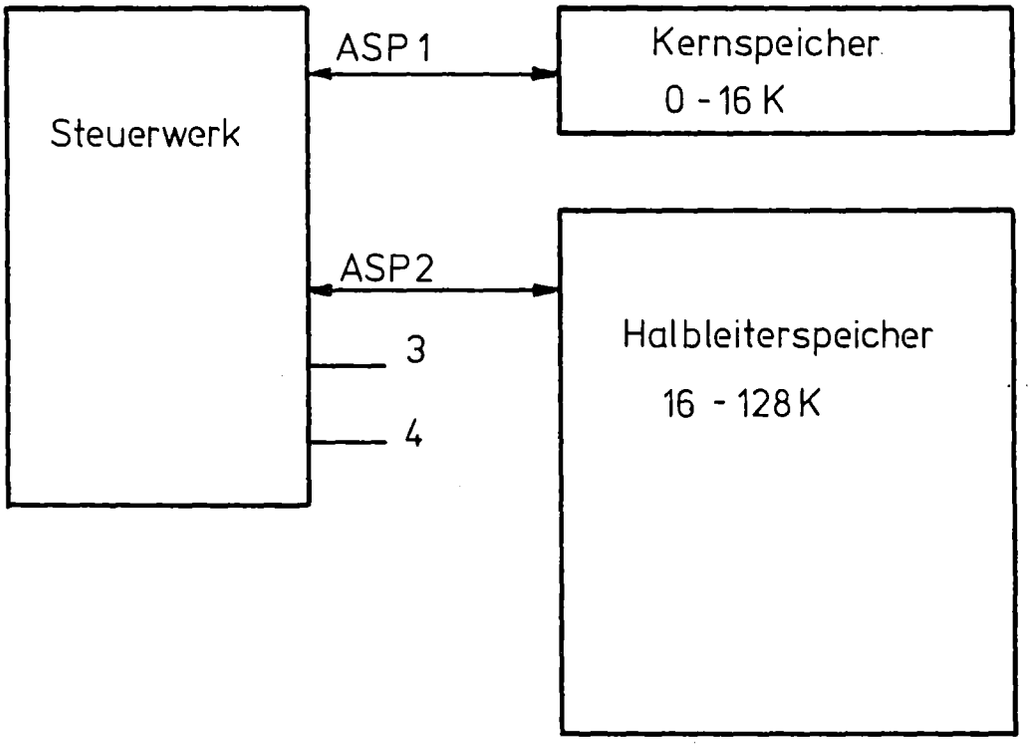


Abb. 2 Speichererweiterung an beliebigem ASP -Kanal

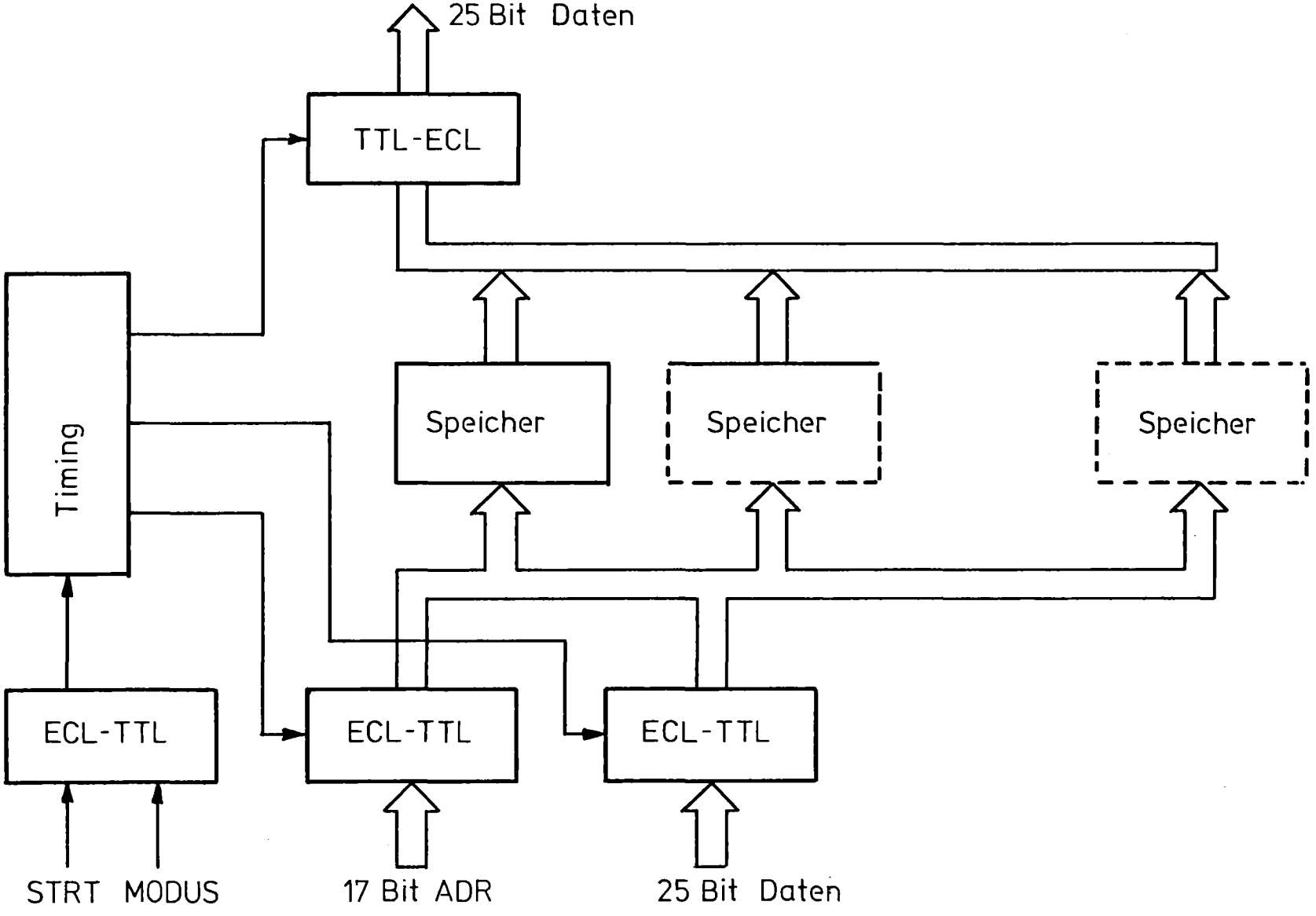


Abb. 3 Blockschaltbild des Halbleiterspeichers

Ein universeller Cross-Assembler für Mikro-Computer

F. Pieper FH Ulm

Zusammenfassung:

Der Cross-Assembler CRAS assembliert auf der S 3o5 Programme für verschiedene Mikro-Computer nach einer einheitlichen Syntax. Z.Zt. (Frühjahr 78) übersetzt CRAS für folgende Mikro-Computer:

KIM I (MCS-65oX von MOS-Technology)
INTEL 8o8o
INTEL 8o48
MOSTEKS F8
SC/MP

Es können beliebig viele weitere Mikro-Computer in CRAS realisiert werden, der Implementierungsaufwand beträgt jeweils 1-5 Tage.
Das CRAS-System wurde mit Hilfe des Metacompilers MECO und des STRACHEY-Makrogenerators STRG in ca. 3 Monaten erstellt.

Inhalt:

Kurzbeschreibung
Anhang: Syntax-Diagramme
Datenflußplan
Liste der Datei- und Programm-Namen
Bedienung
Beispiel-Protokolle

Ein universeller Cross-Assembler für Mikrocomputer

In den verschiedenen Labors der FH Ulm wurde bereits vor einem Jahr mit mehr als 7 verschiedenen Mikroprozessor-Typen experimentiert (eine Standardisierung bzw. 'einheitliche Linie' ist nicht wünschenswert und beim derzeitigen Entwicklungstempo auch nicht vertretbar). Dabei zeigte sich folgender Trend:
Der Aufwand an Zeit, Entwicklungsarbeit und notwendigem Lerneifer steigt für die Software-Herstellung anscheinend noch immer in dem Maße, wie die Mikroprozessoren leistungsfähiger werden, während dieser Aufwand hardware-seitig immer geringer wird (man kann Informatik-Studenten im 5. Semester bereits ohne weiteres einen kompletten Prozeßrechner mit Mikroprozessoren bauen lassen).
Zwar gibt es für alle gängigen Mikrocomputer heute Cross-Assembler, die in einer höheren Programmiersprache geschrieben sind (meist FORTRAN oder BASIC). Sie leiden aber durchweg an folgenden Mängeln:

- sie sind trotz ihrer angeblichen Portabilität nur mit erheblichem Aufwand auf den jeweils verfügbaren Rechnern zu installieren,
- sie sind zu langsam,
- sie folgen in vielen Fällen einer eigenwilligen, jedenfalls nicht immer unmittelbar einsichtigen, häufig unzweckmäßigen Syntax, was stets neuen, unnötigen Lernaufwand bedeutet.

Aus diesen Erfahrungen entstand bald der Wunsch nach einem benutzer-freundlichen, universellen Cross-Assembler-System, welches folgende Eigenschaften haben sollte:

1. Einbettung in das vorhandene RZ-System, um dessen Komfort voll ausnutzen zu können,
2. einheitliche Syntax für alle im Hause eingesetzten Mikrocomputer,
3. möglichst kurze Übersetzungszeiten,
4. möglichst geringer Aufwand bei der Einführung neuer Mikrocomputer.

Zu diesem Wunschkatalog wurde im Sommer 77 ein Konzept entworfen und im Herbst 77 der Cross-Assembler CRAS realisiert.
Der Assembler besteht aus einem Rumpfsegment mit dem Programmnamen CRAS und 4 Overlay-Segmenten, von denen das erste der Übersetzungs-Vorbereitung, das letzte der Code-Ausgabe dient. Die beiden mittleren Segmente besorgen nach dem üblichen 2-Paß-Verfahren die eigentliche Übersetzung (vgl. Datenflußplan).
Zum System gehören ferner die Lade- bzw. List-Programme DALA, MILA, BELI (vgl. Liste der Namen).
Sämtliche Teile dieses Assemblersystems wurden in einer Metasprache formuliert und mit dem zugehörigen Metasprachen-Compiler MECO übersetzt. MECO erzeugt STRACHEY-Makros, die in einem anschließenden Makrogenerator-Lauf STRG in PROSA umgesetzt werden. Auf dieser Ebene wurde auch der gesamte Datenfluß mit äußerst leistungsfähigen Makros zur E/A- und Datei-Organisation programmiert (z.B. Mehrpuffer-Betrieb, automatische Anzeigen- und Ende-Kontrolle, ASCII-Ausgabe, sequentieller und wahlfreier Zugriff mit automatischem paging). Auf der untersten (dritten) Ebene erzeugt schließlich der PROSA-Übersetzer die MC-Teile.
Die 3 Sprachebenen MECO-STRG-PROSA bieten eine äußerst effektive und dabei flexible Implementierungs-Methode.
So sind beispielsweise alle PROSA-Namen bereits in MECO zugänglich und umgekehrt, sodaß man nach Belieben die Sprachebenen wechseln kann. Das generierte MC ist in Bezug auf den Datenfluß praktisch auf Anhieb fehlerfrei, sodaß sich die Fehlersuche auf das Auffinden logischer Fehler auf MECO-Ebene beschränkt. Bei richtiger Anwendung dokumentiert sich MECO selbst.
Mit Hilfe des MECO-STRG-Systems, das der FH Ulm leider erst seit Frühjahr 77 zur Verfügung steht, konnte das Cross-Assembler-System CRAS in weniger als 3 Monaten implementiert werden.
Der eingangs aufgestellte Wunschkatalog wird von CRAS in folgender Weise erfüllt:

ad 1: CRAS läuft auf der 3o5 im ORG I (an der FH Ulm integriert in ein spezielles Monitorsystem) und nutzt die schnelle Peripherie (LKE, SDA, LSA, PS - s. Datenflußplan) voll aus. CRAS-Programme können auf PS gehalten und dort mit den üblichen Methoden editiert werden.

ad 2: Die CRAS-Syntax lehnt sich eng an einige schon vorhandene Assembler, die bereits weit verbreitet und auch durch einige Literatur bekannt sind. Die genaue Beschreibung findet man in den Syntaxdiagrammen, die nach N.WIRTHs Methode erstellt wurden.
Es bereitet keine Schwierigkeiten, die unterschiedlichen Befehlslisten in CRAS zu integrieren. Was die Befehlsdarstellung betrifft, muß man nicht einmal die vom jeweiligen Hersteller definierte Darstellung ändern.
Es genügt, für jeden Befehl einer Befehlsliste folgende Information bereitzuhalten:

1. Mnemo-Code (max. 4 Zeichen)
2. Kennziffer für die Adressierungsart
   (z.Zt. 54 Adressierungsarten)
3. Operation Code
4. Befehlslänge in Bytes.

Eine solche universelle Syntax muß sich natürlich an das 'klassische' mnemonische Schema halten; ein so komfortabler Assembler wie AS 3oo wäre nicht geeignet.

ad 3: Folgende Übersetzungszeiten wurden gemessen:

Minimum: 15 sec (5o Karten, KIM)
Maximum: 17o sec (94o Karten,INTEL8o8o)

(die Zeiten gelten einschließlich Protokoll- und LS-Ausgabe bei Eingabe von Platte).
Damit ist CRAS bis zu 1omal schneller als vergleichbare in BASIC oder FORTRAN geschriebene Cross-Assembler.

ad 4: Einen neuen Mikrocomputer in CRAS zu integrieren, erfordert folgende Schritte:

1. Analyse der Adressierungsarten: sind sie in CRAS schon bekannt oder müssen neue Adressierungsarten definiert werden ?
2. Generierung eines neuen DL 1, der genau die Adressierungsarten des neuen Mikrocomputers

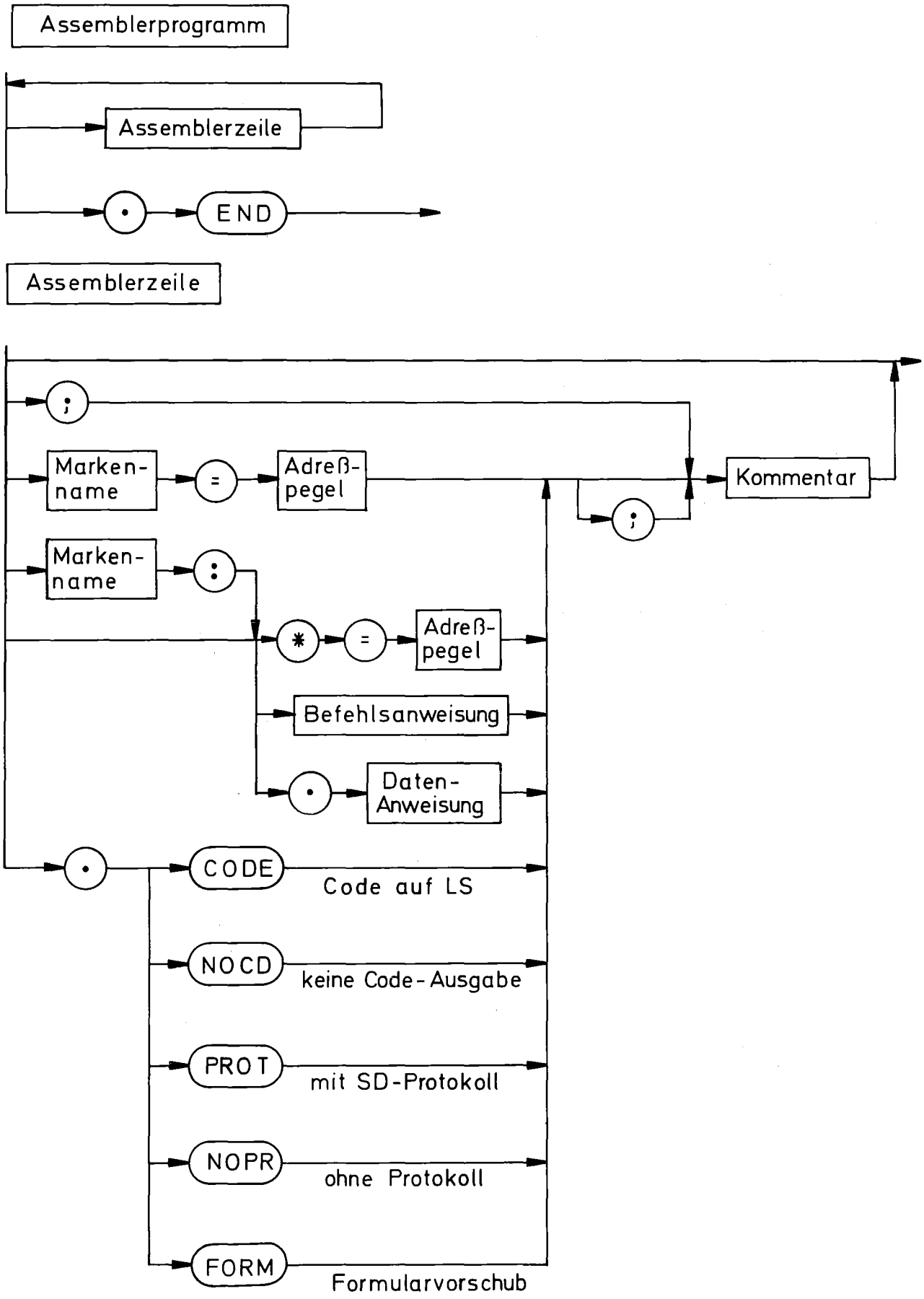
enthält, mit MECO und STRG aus dem System-Masterstapel. Dabei wird der individuelle Kennbuchstabe festgelegt (s. Liste der Namen). Der einzige Grund für dieses Verfahren ist Platz- und Zeit-Optimierung.

3. Codierung der Befehlsliste. Codetabelle und Fehlerliste können ebenfalls individuell definiert werden, sind aber mit hoher Wahrscheinlichkeit identisch mit den schon eingeführten.
4. Laden der individuellen Liste(n) mit DALA, Bekanntgabe der Listennamen, des Kennbuchstabens und der Mikro-Kennung an CRAS mit MILA (s. Bedienung).
5. Drucken der Befehlsliste für die Benutzer mit BELI.

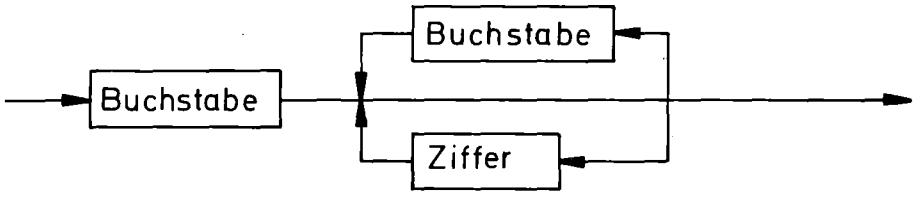
Ein Übersetzungstest ist spätestens beim dritten Versuch erfolgreich, d.h. 'empirisch fehlerfrei' (weitere Fehler wären mit hoher Wahrscheinlichkeit allen Varianten gemeinsam).
Der Aufwand für diese Maßnahmen beträgt 5-2o Stunden Schreibtischarbeit, sowie 4o Minuten Rechenzeit pro Generierlauf.

Abschließend sei darauf hingewiesen, daß bei einer Bewertung dieses Systems bzw. bei einem Vergleich mit anderen Systemen natürlich der 'Heimvorteil' zu beachten ist:
Zum einen ist das System für die eigenen Bedürfnisse maßgeschneidert, zum anderen ist es nur von intimen Kennern mit Minimalaufwand zu erweitern (die o.g. Zeiten wären vielleicht zu verdreifachen, wenn jemand MECO, aber nicht CRAS kennt, sie sind nicht kalkulierbar, wenn jemand beide nicht kennt).
Allerdings dürften die Bedürfnisse an der FH Ulm die gleichen sein wie an vielen anderen Instituten, und für den Autor hat sich der bei MECO und STRG eingesetzte Lernaufwand weit effektvoller ausgezahlt als bei jedem anderen Programmiersystem.

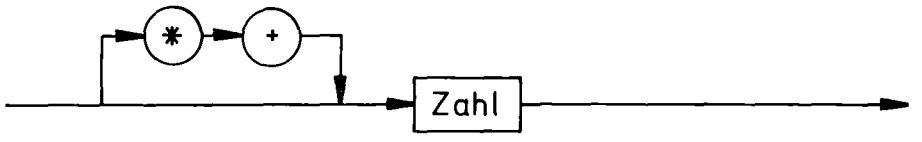
305 - CROSS - ASSEMBLER CRAS



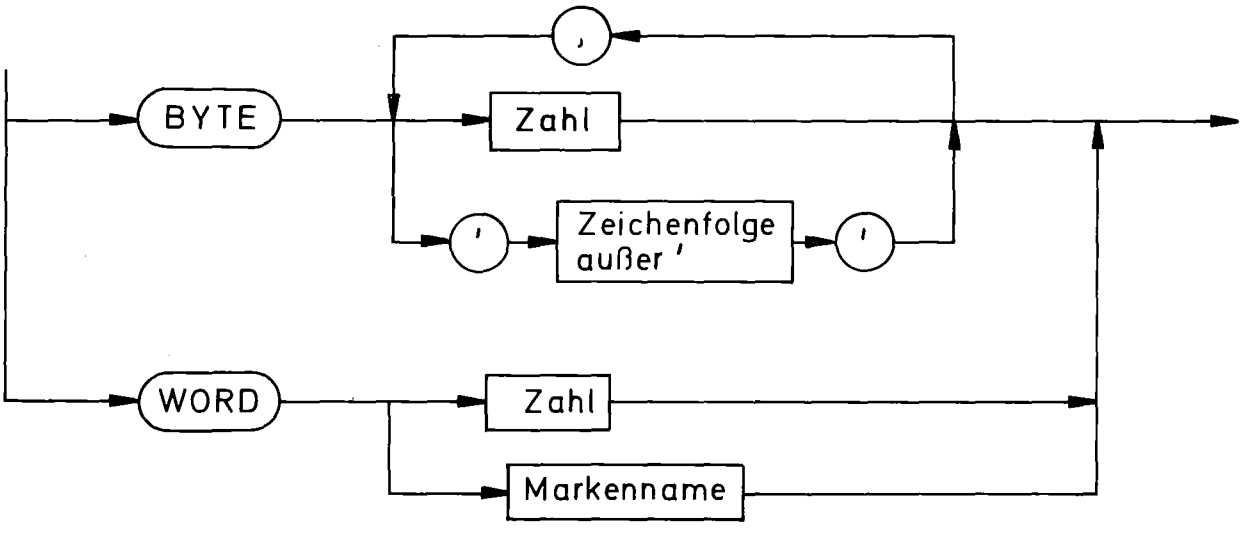
Markenname



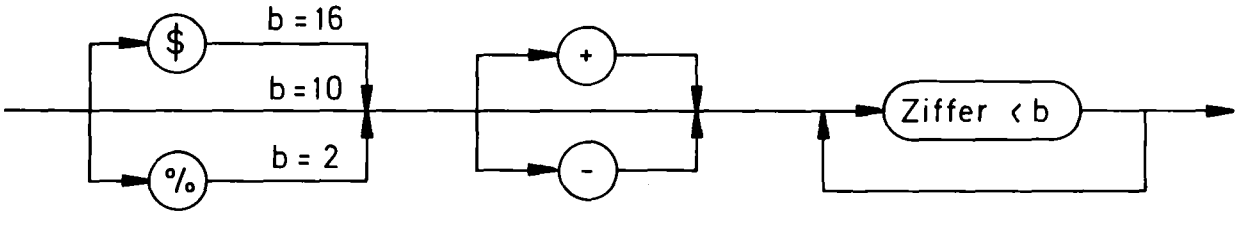
Adreßpegel



Datenanweisung

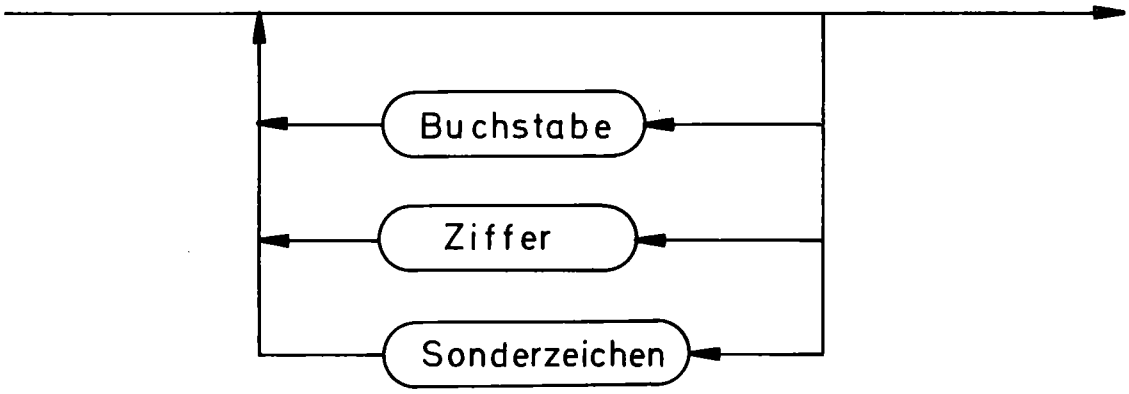


Zahl



Zeichenfolge

Kommentar



Befehlsanweisung — vgl. Befehlsliste für den jeweiligen Mikrocomputer

Zeichenvorrat: Eingabe: LK 26er Code (305-Norm)

Ausgabe: LS ASCII entsprechend der jeweiligen Normierung.

Buchstaben: A, ...,Z

Ziffern: Ø,..., 9, A,...,F

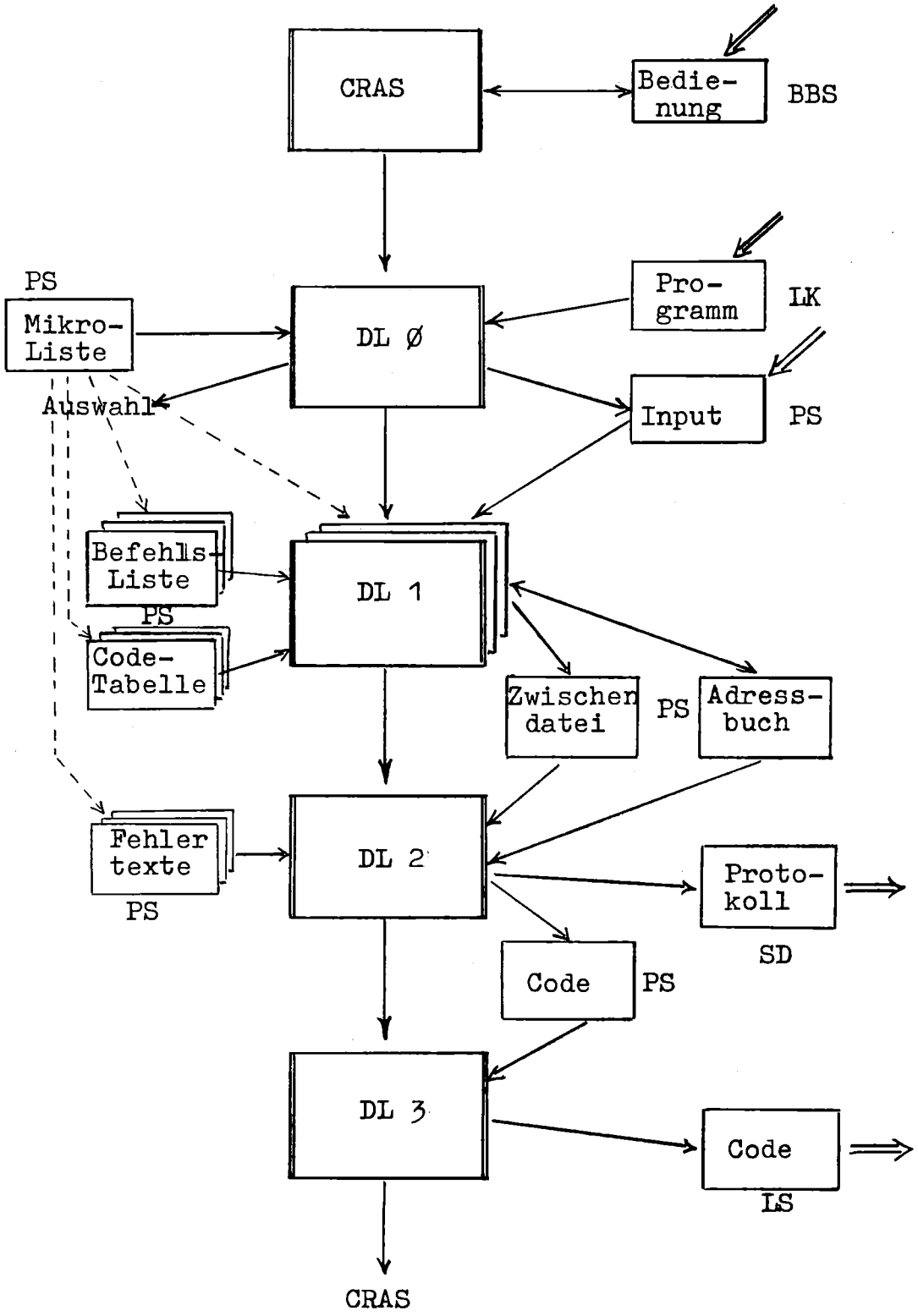
Sonderzeichen: +-/*,.'=:;$%()#

Anmerkung: Das ASCII-Zeichen "@" wird im 26er Code ersetzt durch " * ".

Die Zeichen; % # werden für die SD-Ausgabe (und nur dort) geändert.

3o5 - CROSS ASSEMBLER CRAS

Datenfluß



3o5 - CROSS ASSEMBLER CRAS

Programm- und Datei-Namen

1. Assembler

Programm-Namen (3o5 MC): CRAS, CRAØ, CR.1, CRA2, CRA3
Arbeitsform: +RAØ, +R.1, +RA2, +RA3
"." ist durch den Kennbuchstaben (s. Mikroliste) zu ersetzen (bislang nur für DL 1).
Die Dateien der Arbeitsform sind bei Einführung einer neuen Version zu löschen.

2. Scratch-Dateien

Input bei LK-Eingabe: +INP (bei PS-Eingabe: frei, permanent)
Zwischendatei: ++ZW
Adreßbuch: ++AD
Code: ++CD
Die Lebensdauer dieser Dateien beträgt jeweils nur ein Übersetzungslauf.

3. Auswahl-Dateien

Für jeden Mikrocomputer gibt es drei individuelle Dateien, mit denen spezielle Übersetzer-Eigenschaften ausgewählt werden. Ihre symbolischen Namen sind:
Befehlsliste: BELI
Codetabelle: COTA (bislang nur ASCII)
Fehlerliste: FELI (bislang Einheitsliste)
Zu jedem symbolischen Namen darf es beliebig viele physikalische Namen geben.

4. Mikroliste

Die Mikroliste (Dateiname +MIL) enthält für jeden Mikrocomputer einen Satz mit folgender Information:
physikalischer Name von BELI
" " " COTA
" " " FELI
Kennbuchstabe zur Auswahl des spezifischen DL 1. Dieser Parameter ist als einziger fest vorgegeben.
maximal 12 Zeichen zur Identifikation des jeweiligen Mikrocomputers (Bedienung CRAS).

5. Hilfsprogramme

DALA, MILA, BELI (s."Bedienung").

305 - CROSS ASSEMBLER CRAS

Bedienung

1. CRAS: CRAS=LK,Kennung; Quellprogramm auf LK
CRAS=PS=NAME,Kennung; Quellprogramm in PS-Datei NAME
Kennung: max. 12 Zeichen, die den gewünschten Mikrocomputer identifizieren (s. Mikroliste).

2. DALA: Laden der Auswahldateien von LK. Die Lochkarten stehen dem RZ im richtigen Format zur Verfügung.
DALA=BELI,NAME; NAME ist jeweils der
DALA=COTA,NAME; gewünschte physikalische
DALA=FELI,NAME; Dateiname.
DALA=MILI; Einrichten Mikroliste.

3. MILA: Mikroliste laden, d.h. Bekanntgabe eines weiteren Mikrocomputers an CRAS.
MILA=beli,cota,feli,kennbuchstabe,kennung;
physikalische Dateinamen!
Anmerkung: Obwohl die eindeutige Auswahl der richtigen Übersetzer-Eigenschaften durch den Kennbuchstaben gesteuert wird, ist der Suchschlüssel in der Mikroliste die Kennung. Dies ermöglicht die Einrichtung verschiedener Kennungen bei identischen Eigenschaften.
MILA=LIST; Ausdrucken der Mikroliste.

4. BELI: Ausdrucken einer Befehlsliste (Handliste für den Benutzer).
BELI=BELO,NAME; nach Operationcode aufsteigend sortiert.
BELI=BETA,NAME; als (16,16)-Tabelle. NAME ist ein physikalischer BELI-Name.
BELI=LIST; Ausdrucken aller Adressierungsmodi.

```
BRPS-2,2000,BELI;;BR;STRT-2;;

   2 BELI-V1 1    PE 7201

BELI:   GIB BELI-BETA/BELO,NAME
BELI'20.3.78;;
BELI:   GIB BELI-BETA/BELO,NAME
BELI-BETA,+KIM;;
BELI:   GIB BELI-BETA/BELO,NAME
BELI-BETA,+IN8;;
BELI:   GIB BELI-BETA/BELO,NAME
BELI-BETA,+SCP;;
BELI:   GIB BELI-BETA/BELO,NAME
BELI-BETA,+IN4;;
BELI:   GIB BELI-BETA/BELO,NAME
BELI-BETA,+MF8;;
BELI:   GIB BELI-BETA/BELO,NAME
BELI-BELO,+IN8;;
BELI:   GIB BELI-BETA/BELO,NAME
BELI;;

  2 BELI-V1 1   ENDE

BRPS-2,2000,MILA;;BR;STRT-2;;

   2 MILA-V1 1    PE 3441

MILA:
MILA-LIST;;
MILA:     BEFEHLS-  CODE-  FEHLER-   KENN-   MIKRO-
MILA:      LISTE   TABELLE  LISTE   BUCHST.  KENNUNG

MILA:       +KIM    +COT    +FEL      A     KIM
MILA:       +IN4    +COT    +FEL      D     INTEL8048
MILA:       +IN8    +COT    +FEL      B     INTEL8080
MILA:       +SCP    +COT    +FEL      C     SC/MP
MILA:       +MF8    +COT    +FEL      E     MOSTEKS-F8
MILA:

MILA:
MILA;;

  2 MILA-V1 1   ENDE

BRPS-2,2000,CRAS;;BR;STRT-2;;

   2 CRAS-V1 1    PE 8614

CRAS:   GIB CRAS-QUELLE,MIKRO
CRAS'20.3.78;;
CRAS:   GIB CRAS-QUELLE,MIKRO
CRAS-PS-TINT,INTEL8080;;
CRAS:   GIB CRAS-QUELLE,MIKRO
CRAS;;

  2 CRAS-V1 1   ENDE
```

```
                    S I E M E N S  S Y S T E M  3 0 0

KIM          CROSS-ASSEMBLER:    TEST     DATUM:  20.3.78              BLATT:     1


   1                                                                  AUSGABEPROGRAMM
   2
   3                                IVEC     =     $17FA              FINTERRUPTVEKTOR
   4                                A        =     $F1
   5                                OUTCH    =     $1EA0              ZEICHENAUSGABE
   6                                AU       =     $1C00
   7  0000     16                   ANZ:     .BYTE 22
   8  0001                          ANFADR:  *=    *+2
   9  0003                          SAVEY:   *=    *+2
  10  0005                          BYTL:    *=    *+1
  11                                         *=    $200
  12  0200     AD 14 02             ANF      LDA   ANF               ANFANGSADRESSE LOW
  13  0203     85 01                         STA   ANFADR            ANFANGSADRESSE TEXT LOW
  14  0205     AD 15 02                      LDA   ANF+1             ANFANGSADRESSE HIGH
  15  0208     85 02                         STA   ANFADR+1          ANFANGSADRESSE TEXT HIGH
  16  020A     A5 00                         LDA   ANZ
  17  020C     85 05                         STA   BYTL              ANZAHL AUSZUGEBENDER BYTEZ
  18  020E     20 30 02                      JSR   BSAU              UNTERPROGRRAMM
  19  0211     4C 00 1C                      JMP   AU
  20  0214     16 02                ANF:     .WORD TOP
  21  0216     0A 0D 57 4F          TOP:     .BYTE $A,$D,'WO BLEIBT DER'
      021A     20 42 4C 45
      021E     49 42 54 20
      0222     44 45 52
  22  0225     20 53 45 4B                   .BYTE ' SEKT',$D,$A
      0229     54 0D 0A
  23                                         *=    $230
  24  0230     A9 00                BSAU:    LDA   #0
  25  0232     8D FA 17                      STA   IVEC
  26  0235     85 F1                         STA   A
  27  0237     A9 1C                         LDA   #$1C
  28  0239     8D FB 17                      STA   IVEC+1
  29  023C     A0 00                         LDY   #0
  30  023E     B1 01                AUSG:    LDA   (ANFADR),Y
  31  0240     84 03                         STY   SAVEY
  32  0242     20 A0 1E                      JSR   OUTCH             AUSGABE
  33  0245     A4 03                         LDY   SAVEY
```

```
                    S I E M E N S  S Y S T E M  3 0 0

KIM          CROSS-ASSEMBLER:    TEST     DATUM:  20.3.78              BLATT:     2


  34  0247     C8                            INY
  35  0248     C4 05                         CPY   BYTL
  36  024A     90 F2                         BCC   AUSG
  37  024C     60                            RTS
  38                                         .END
```

S I E M E N S S Y S T E M 3 0 0

INTEL8080 CROSS-ASSEMBLER 04 TINT DATUM: 22.6.78 BLATT: 1

```
   1
   2                                    *=    $1000
   3    1000    31 E1 13                LXI   SP,SPPOOL       STACKPOINTER LADEN
   4    1003    3E 81                   MVI   A,$81            8255 PROGRAMMIEREN
   5    1005    D3 F7                   OUT   $F7              AUSGABE IN STATUSREGISTER DES 8255
   6    1007    3E F8                   MVI   A,/11111000      SCHRITTMOTOR DEFINIERT SETZEN
   7    1009    D3 F5                   OUT   $F5              AUSGABE PORT B
   8    100B    CD 34 11                CALL  WAIT
   9    100E    3E FC                   MVI   A, /11111100
  10    1010    D3 F5                   OUT   $F5
  11    1012    CD 34 11                CALL  WAIT
  12    1015    1E 00                   MVI   E,0              BITMUSTERPOINTER SETZEN
  13    1017    21 93 10        ANLAUF: LXI   H,MASK1
  14    101A    16 00                   MVI   D,00
  15    101C    19                      DAD   D
  16    101D    DB F6                   IN    $F6
  17    101F    E6 01                   ANI   $01
  18    1021    C2 36 10                JNZ   GRUND
  19    1024    7E                      MOV   A,M
  20    1025    D3 F5                   OUT   $F5
  21    1027    CD 34 11                CALL  WAIT
  22    102A    1C                      INR   E
  23    102B    3E 08                   MVI   A,8
  24    102D    BB                      CMP   E
  25    102E    C2 17 10                JNZ   ANLAUF
  26    1031    1E 00                   MVI   E,0
  27    1033    C3 17 10                JMP   ANLAUF
  28    1036    21 92 10        GRUND:  LXI   H,LZEICH         ADR DES DRUCKZEICHENSPEICHERS LADEN
  29    1039    36 35                   MVI   M,$35            MOTOR SITZT AUF GRUNDSTELLUNG
  30
  31    103B    21 A3 10        UNTAET: LXI   H,HASH
  32    103E    CD 3B 11                CALL  CI
  33    1041    16 21                   MVI   D,$21            KONTROLLE OB SONDERZEICHEN,NIEDERWERTIG
  34    1043    BA                      CMP   D
  35    1044    FA 3B 10                JM    UNTAET           BEI SONDERZEICHEN ZURUECK
  36    1047    16 7F                   MVI   D,$7F            KONTROLLE OB SONDERZEICHEN,HOEHERWERTIG
  37    1049    BA                      CMP   D
```

S I E M E N S S Y S T E M 3 0 0

INTEL8080 CROSS-ASSEMBLER 04 TINT DATUM: 22.6.78 BLATT: 2

```
  38    104A    F2 3B 10                JP    UNTAET           BEI SONDERZEICHEN ZURUECK
  39    104D    0E 21                   MVI   C,$21
  40    104F    91                      SUB   C
  41    1050    4B                      MOV   C,E
  42    1051    16 00                   MVI   D,00
  43    1053    5F                      MOV   E,A
  44    1054    19                      DAD   D
  45    1055    59                      MOV   E,C
  46    1056    4E                      MOV   C,M
  47    1057    21 92 10                LXI   H,LZEICH
  48    105A    7E                      MOV   A,M              DIFFERENZ ZWISCHEN ALTEM UND NEUEN
  49    105B    91                      SUB   C                ZEICHEN BERECHNEN
  50    105C    CA 3B 10                JZ    UNTAET           ALTES = NEUES ZEICHEN
  51    105F    F2 7A 10                JP    LINS             DIFF. POSITIV  = LINKSDREHUNG
  52    1062    16 D0                   MVI   D,-48            DIFFERENZ GROESSER 48 SCHRITTE
  53    1064    BA                      CMP   D
  54    1065    FA 74 10                JM    RL               JA,DREHRICHTUNGSWECHSEL RECHTS-LINKS
  55    1068    CD 17 11        RE:     CALL  RECHTS           POSITIONIEREN IN RECHTSRICHTUNG
  56    106B    21 93 10                LXI   H,MASK1          ADR DES EXAKTEN BITMUSTERS LADEN
  57    106E    CD 2D 11                CALL  PORT1            SCHRITTMOTOR EXAKT AUF ZEICHEN SETZEN
  58    1071    C3 3B 10                JMP   UNTAET
  59    1074    16 60           RL:     MVI   D,96             SCHRITTE LNKS=SCHRITTE UMFANG-SCHRITTE REC
  60    1076    82                      ADD   D
  61    1077    C3 80 10                JMP   LI               SPRINGE DREHRICHTUNG LINKS
  62    107A    16 30           LINS:   MVI   D,48             DIFFERENZ GROESSER 48 SCHRITTE
  63    107C    BA                      CMP   D
  64    107D    F2 8C 10                JP    LR               JA,DREHRICHTUNGSWECHSEL LINKS-RECHTS
  65    1080    CD 01 11        LI:     CALL  LINKS            POSITIONIEREN IN LINKSRICHTUNG
  66    1083    21 93 10                LXI   H,MASK1          ADR DES EXAKTEN BITMUSTERS LADEN
  67    1086    CD 2D 11                CALL  PORT1            SCHRITTMOTOR EXAKT AUF ZEICHEN SETZEN
  68    1089    C3 3B 10                JMP   UNTAET
  69    108C    16 A0           LR:     MVI   D,-96            SCHRITTE RECHTS=SCHRITTE UMFANG-SCHRITTE L
  70    108E    82                      ADD   D
  71    108F    C3 68 10                JMP   RE               SPRINGE DREHRICHTUNG RECHTS
  72
  73                            SPPOOL  =     $13E1            ADR DES STACKPOINTERPOOLS
  74    1092    00              LZEICH: .BYTE 0                DRUCKZEICHENSPEICHER
```

S I E M E N S S Y S T E M 3 0 0

BEF. LISTE: INTEL8048 DATEI: +IN4 DATUM: 22.6.78 BLATT: 10

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP		OUTL	ADD	JMP	ENI		DEC	INS	IN	IN		MOVD	MOVD	MOVD	MOVD
1	INC	INC	JBO	ADDC	CALL	DIS	JTF	INC	INC	INC	INC	INC	INC	INC	INC	INC
2	XCH	XCH		MOV	JMP	ENI	JNTO	CLR	XCH	XCH	XCH	XCH	XCH	XCH	XCH	XCH
3	XCHD	XCHD	JB1		CALL	DIS	JTO	CPL	MOVD	MOVD	OUTL					
4	ORL	ORL	MOV	ORL	JMP	STRT	JNT1	SWAP	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL
5	ANL		JB2	ANL	CALL	STRT	JT1	DA	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL
6	ADD	ADD	MOV		JMP	STOP		RRC	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD
7	ADDC	ADDC	JB3		CALL	ENTO	JF1	RR	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC
8	MOVX	MOVX		RET	JMP	CLR	JNI	ORLD	ORL	ORL	ORL					
9	MOVX	MOVX	JB4	RETR	CALL	CPL	JNZ	CLR	ANL	ANL	ANL		ANLD	ANLD	ANLD	ANLD
A	MOV	MOV		MOVP	JMP	CLR		CPL	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
B	MOV	MOV	JB5	JMPP	CALL	CPL	JFO	MOV								
C					JMP		JZ		DEC	DEC	DEC	DEC	DEC	DEC	DEC	DEC
D	XRL	XRL	JB6	XRL	CALL	SEL	SEL	MOV	SEL	XRL	XRL	XRL	XRL	XRL	XRL	XRL
E				MOV3	JMP		JNC	RL	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ
F	MOV	MOV	JB7		CALL		JC	RLC	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV

S I E M E N S S Y S T E M 3 0 0

BEF. LISTE: INTEL8080 DATEI: +IN8 DATUM: 22.6.78 BLATT: 5

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LXI	STAX	INX	INR	DCR	MVI	RLC		DAD	LDAX	DCX	INR	DCR	MVI	RRC
1		LXI	STAX	INX	INR	DCR	MVI	RAL		DAD	LDAX	DCX	INR	DCR	MVI	RAR
2		LXI	SHLD	INX	INR	DCR	MVI	DAA		DAD	LHLD	DCX	INR	DCR	MVI	CMA
3		LXI	STA	INX	INR	DCR	MVI	STC		DAD	LDA	DCX	INR	DCR	MVI	CMC
4	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
5	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
6	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
7	MOV	MOV	MOV	MOV	MOV	MOV	HLT	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
8	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADC	ADC	ADC	ADC	ADC	ADC	ADC	ADC
9	SUB	SUB	SUB	SUB	SUB	SUB	SUB	SUB	SBB	SBB	SBB	SBB	SBB	SBB	SBB	SBB
A	ANA	ANA	ANA	ANA	ANA	ANA	ANA	ANA	XRA	XRA	XRA	XRA	XRA	XRA	XRA	XRA
B	ORA	ORA	ORA	ORA	ORA	ORA	ORA	ORA	CMP	CMP	CMP	CMP	CMP	CMP	CMP	CMP
C	RNZ	POP	JNZ	JMP	CNZ	PUSH	ADI	RST	RZ	RET	JZ		CZ	CALL	ACI	RST
D	RNC	POP	JNC	OUT	CNC	PUSH	SUI	RST	RC		JC	IN	CC		SBI	RST
E	RPO	POP	JPO	XTHL	CPO	PUSH	ANI	RST	RPE	PCHL	JPE	XCHG	CPE		XRI	RST
F	RP	POP	JP	DI	CP	PUSH	ORI	RST	RM	SPHL	JM	EI	CM		CPI	RST

S I E M E N S S Y S T E M 3 0 0

BEF. LISTE: KIM DATEI: +KIM DATUM: 22.6.78 BLATT: 14

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BRK	ORA				ORA	ASL		PHP	ORA	ASL			ORA	ASL	
1	BPL	ORA				ORA	ASL		CLC	ORA				ORA	ASL	
2	JSR	AND			BIT	AND	ROL		PLP	AND	ROL		BIT	AND	ROL	
3	BMI	AND				AND	ROL		SEC	AND				AND	ROL	
4	RTI	EOR				EOR	LSR		PHA	EOR	LSR		JMP	EOR	LSR	
5	BVC	EOR				EOR	LSR		CLI	EOR				EOR	LSR	
6	RTS	ADC				ADC	ROR		PLA	ADC	ROR		JMP	ADC	ROR	
7	BVS	ADC				ADC	ROR		SEI	ADC				ADC	ROR	
8		STA			STY	STA	STX		DEY		TXA		STY	STA	STX	
9	BCC	STA			STY	STA	STX		TYA	STA	TXS			STA		
A	LDY	LDA	LDX		LDY	LDA	LDX		TAY	LDA	TAX		LDY	LDA	LDX	
B	BCS	LDA			LDY	LDA	LDX		CLV	LDA	TSX		LDY	LDA	LDX	
C	CPY	CMP			CPY	CMP	DEC		INY	CMP	DEX		CPY	CMP	DEC	
D	BNE	CMP				CMP	DEC		CLD	CMP				CMP	DEC	
E	CPX	SBC			CPX	SBC	INC		INX	SBC	NOP		CPX	SBC	INC	
F	BEQ	SBC				SBC	INC		SED	SBC				SBC	INC	

S I E M E N S S Y S T E M 3 0 0

BEF. LISTE: MOSTEKS-F8 DATEI: +MF8 DATUM: 22.6.78 BLATT: 19

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR		LR	LR	LR
1	LR	LR	SR	SL	SR	SL	LM	ST	COM	LNK	DI	EI	POP	LR	LR	INC
2	LI	NI	OI	XI	AI	CI	IN	OUT	PI	JMP	DCI	NOP	XDC			
3	DS	DS	DS	DS	DS	DS	DS	DS	DS	DS	DS	DS	DS	DS	DS	
4	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	
5	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	LR	
6	LISU	LISU	LISU	LISU	LISU	LISU	LISU	LISU	LISL	LISL	LISL	LISL	LISL	LISL	LISL	LISL
7	CLR	LIS	LIS	LIS	LIS	LIS	LIS	LIS	LIS	LIS	LIS	LIS	LIS	LIS	LIS	LIS
8		BP	BC		BZ				AM	AMD	NM	OM	XM	CM	ADC	BR7
9	BR	BM	BNC		BNZ				BNO							
A	INS	INS	INS	INS	INS	INS	INS	INS	INS	INS	INS	INS	INS	INS	INS	INS
B	OUTS	OUTS	OUTS	OUTS	OUTS	OUTS	OUTS	OUTS	OUTS	OUTS	OUTS	OUTS	OUTS	OUTS	OUTS	OUTS
C	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	AS	
D	ASD	ASD	ASD	ASD	ASD	ASD	ASD	ASD	ASD	ASD	ASD	ASD	ASD	ASD	ASD	
E	XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	
F	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	

S I E M E N S S Y S T E M 3 0 0

BEF. LISTE: SC/MP DATEI: +SCP DATUM: 22.6.78 BLATT: 22

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	HALT	XAE	CCL	SCL	DINT	IEN	CSA	CAS	NOP							
1										SIO			SR	SRL	RR	RRL
2																
3	XPAL	XPAL	XPAL	XPAL	XPAH	XPAH	XPAH	XPAH					XPPC	XPPC	XPPC	XPPC
4	LDE															
5	ANE								ORE							
6	XRE								DAE							
7	ADE								CAE							
8																DLY
9	JMP	JMP	JMP	JMP	JP	JP	JP	JP	JZ	JZ	JZ	JZ	JNZ	JNZ	JNZ	JNZ
A									ILD	ILD	ILD	ILD				
B									DLD	DLD	DLD	DLD				
C	LD	LD	LD	LD	LDI	LD	LD	LD	ST	ST	ST	ST		ST	ST	ST
D	AND	AND	AND	AND	ANI	AND	AND	AND	OR	OR	OR	OR	ORI	OR	OR	OR
E	XOR	XOR	XOR	XOR	XRI	XOR	XOR	XOR	DAD	DAD	DAD	DAD	DAI	DAD	DAD	DAD
F	ADD	ADD	ADD	ADD	ADI	ADD	ADD	ADD	CAD	CAD	CAD	CAD	CAI	CAD	CAD	CAD

S I E M E N S S Y S T E M 3 0 0

BEF. LISTE: INTEL8048 DATEI: +IN4 DATUM: 22.6.78 BLATT: 6

DIE ZAHLEN IN KLAMMERN GEBEN BITPOSITIONEN IM OPERATIONCODE AN (0-7 ENTSPRECHEND DER DUALWERT-STELLE)

OPCODE	MNEMONIC	ADDRESSING MODE	BYTES	OPCODE	MNEMONIC	ADDRESSING MODE	BYTES
00	NOP	IMPLIED	1	02	OUTL	BUS,A	1
03	ADD	A,IMMEDIATE	2	04	JMP	ADDR.HIGH(5-7)	2
05	ENI	INTERRUPT(5)	1	07	DEC	ACCUMULATOR	1
08	INS	IMPLIED	1	09	IN	A,PORT1,2(0,1)	1
0A	IN	A,PORT1,2(0,1)	1	0C	MOVD	A,PORT4-7(0,1)	1
0D	MOVD	A,PORT4-7(0,1)	1	0E	MOVD	A,PORT4-7(0,1)	1
0F	MOVD	A,PORT4-7(0,1)	1	10	INC	REG.INDIRECT(0)	1
11	INC	REG.INDIRECT(0)	1	12	JB0	ABSOLUTE ADDR.	2
13	ADDC	A,IMMEDIATE	2	14	CALL	ADDR.HIGH(5-7)	2
15	DIS	INTERRUPT(5)	1	16	JTF	ABSOLUTE ADDR.	2
17	INC	ACCUMULATOR	1	18	INC	REGISTER(0-2)	1
19	INC	REGISTER(0-2)	1	1A	INC	REGISTER(0-2)	1
1B	INC	REGISTER(0-2)	1	1C	INC	REGISTER(0-2)	1
1D	INC	REGISTER(0-2)	1	1E	INC	REGISTER(0-2)	1
1F	INC	REGISTER(0-2)	1	20	XCH	A,REG.IND.(0)	1
21	XCH	A,REG.IND.(0)	1	23	MOV	A,IMMEDIATE	2
24	JMP	ADDR.HIGH(5-7)	2	25	ENI	INTERRUPT(5)	1
26	JNTO	ABSOLUTE ADDR.	2	27	CLR	A/CARRY/FLAG	1
28	XCH	A,REGISTER(0-2)	1	29	XCH	A,REGISTER(0-2)	1
2A	XCH	A,REGISTER(0-2)	1	2B	XCH	A,REGISTER(0-2)	1
2C	XCH	A,REGISTER(0-2)	1	2D	XCH	A,REGISTER(0-2)	1
2E	XCH	A,REGISTER(0-2)	1	2F	XCH	A,REGISTER(0-2)	1
30	XCHD	A,REG.IND.(0)	1	31	XCHD	A,REG.IND.(0)	1
32	JB1	ABSOLUTE ADDR.	2	34	CALL	ADDR.HIGH(5-7)	2
35	DIS	INTERRUPT(5)	1	36	JT0	ABSOLUTE ADDR.	2
36	MOVD	PORT4-7(0,1),A	1	37	CPL	A/CARRY/FLAG	1
37	MOVD	PORT4-7(0,1),A	1	38	MOVD	PORT4-7(0,1),A	1
39	MOVD	PORT4-7(0,1),A	1	39	OUTL	PORT1,2(0,1),A	1
3A	OUTL	PORT1,2(0,1),A	1	40	ORL	A,REG.IND.(0)	1
41	ORL	A,REG.IND.(0)	1	42	MOV	A,ST.WORD/TIMER	1
43	ORL	A,IMMEDIATE	2	44	JMP	ADDR.HIGH(5-7)	2
45	STRT	START(4)	1	46	JNT1	ABSOLUTE ADDR.	2
47	SWAP	IMPLIED	1	48	ORL	A,REGISTER(0-2)	1

S I E M E N S S Y S T E M 3 0 0

BEF. LISTE: INTEL8048 DATEI: +IN4 DATUM: 22.6.78 BLATT: 7

49	ORL	A,REGISTER(0-2)	1	4A	ORL	A,REGISTER(0-2)	1
4B	ORL	A,REGISTER(0-2)	1	4C	ORL	A,REGISTER(0-2)	1
4D	ORL	A,REGISTER(0-2)	1	4E	ORL	A,REGISTER(0-2)	1
4F	ORL	A,REGISTER(0-2)	1	50	ANL	ABSOLUTE ADDR.	1
52	JB2	ABSOLUTE ADDR.	2	53	ANL	A,IMMEDIATE	2
54	CALL	ADDR.HIGH(5-7)	2	55	STRT	START(4)	1
56	JT1	ABSOLUTE ADDR.	2	57	DA	IMPLIED	1
58	ANL	A,REGISTER(0-2)	1	59	ANL	A,REGISTER(0-2)	1
5A	ANL	A,REGISTER(0-2)	1	5B	ANL	A,REGISTER(0-2)	1
5C	ANL	A,REGISTER(0-2)	1	5D	ANL	A,REGISTER(0-2)	1
5E	ANL	A,REGISTER(0-2)	1	5F	ANL	A,REGISTER(0-2)	1
60	ADD	A,REG.IND.(0)	1	61	ADD	A,REG.IND.(0)	1
62	MOV	ST.WORD/TIMER,A	1	64	JMP	ADDR.HIGH(5-7)	2
65	STOP	IMPLIED	1	67	RRC	IMPLIED	1
68	ADD	A,REGISTER(0-2)	1	69	ADD	A,REGISTER(0-2)	1
6A	ADD	A,REGISTER(0-2)	1	6B	ADD	A,REGISTER(0-2)	1
6C	ADD	A,REGISTER(0-2)	1	6D	ADD	A,REGISTER(0-2)	1
6E	ADD	A,REGISTER(0-2)	1	6F	ADD	A,REGISTER(0-2)	1
70	ADDC	A,REG.IND.(0)	1	71	ADDC	A,REG.IND.(0)	1
72	JB3	ABSOLUTE ADDR.	2	74	CALL	ADDR.HIGH(5-7)	2
75	ENT0	IMPLIED	1	76	JF1	ABSOLUTE ADDR.	2
77	RR	IMPLIED	1	78	ADDC	A,REGISTER(0-2)	1
79	ADDC	A,REGISTER(0-2)	1	7A	ADDC	A,REGISTER(0-2)	1
7B	ADDC	A,REGISTER(0-2)	1	7C	ADDC	A,REGISTER(0-2)	1
7D	ADDC	A,REGISTER(0-2)	1	7E	ADDC	A,REGISTER(0-2)	1
7F	ADDC	A,REGISTER(0-2)	1	80	MOVX	A,REG.IND.(0)	1
81	MOVX	A,REG.IND.(0)	1	83	RET	IMPLIED	1
84	JMP	ADDR.HIGH(5-7)	2	85	CLR	A/CARRY/FLAG	1
86	JNI	ABSOLUTE ADDR.	2	86	ORLD	PORT4-7(0,1),A	1
87	ORLD	PORT4-7(0,1),A	1	88	ORL	PORT/BUS,IMMED.	2
88	ORLD	PORT4-7(0,1),A	1	89	ORL	PORT/BUS,IMMED.	2
89	ORLD	PORT4-7(0,1),A	1	8A	ORL	PORT/BUS,IMMED.	2
90	MOVX	REG.INDIRECT,A	1	91	MOVX	REG.INDIRECT,A	1
92	JB4	ABSOLUTE ADDR.	2	93	RETR	IMPLIED	1
94	CALL	ADDR.HIGH(5-7)	2	95	CPL	A/CARRY/FLAG	1
96	JNZ	ABSOLUTE ADDR.	2	97	CLR	A/CARRY/FLAG	1
98	ANL	PORT/BUS,IMMED.	2	99	ANL	PORT/BUS,IMMED.	2

Anschluß eines Trommelspeichers an die Arbeitsspeicher-nahtstelle der S 301

W. Becker, GH. Paderborn

Im Fachbereich 16, Elektrische Energietechnik der Gesamthochschule Paderborn, bestand die Aufgabe, eine Trommel mit einem Speicherplatz von 128 kW an den vorhandenen Prozeßrechner S 301 anzuschließen, der bis dahin über keinen Hintergrundspeicher verfügte. Der Trommelspeicher war uns von der Firma Siemens überlassen worden; Mittel für den Anschluß standen nicht zur Verfügung und da die 301 als nicht mehr ausbauwürdig galt, bestanden auch keine Aussichten, hierfür Mittel zu erhalten. So bestand nur die recht reizvolle Möglichkeit, im Rahmen von Abschlußarbeiten die Trommel an die Arbeitsspeichernahtstelle der S 301 anzuschließen. Wegen des Umfanges der Arbeiten wurde ein Team von 3 Studenten (Barbara Krampe, Meinolf Schlieper, Peter Schütte) gebildet, die den Koppler gemeinsam entwickelten und bauten. Innerhalb des Teams hatte jeder einen Schwerpunkt zu bearbeiten: Gerätesteuerung, Trommelsteuerung, Fehlerbearbeitung.

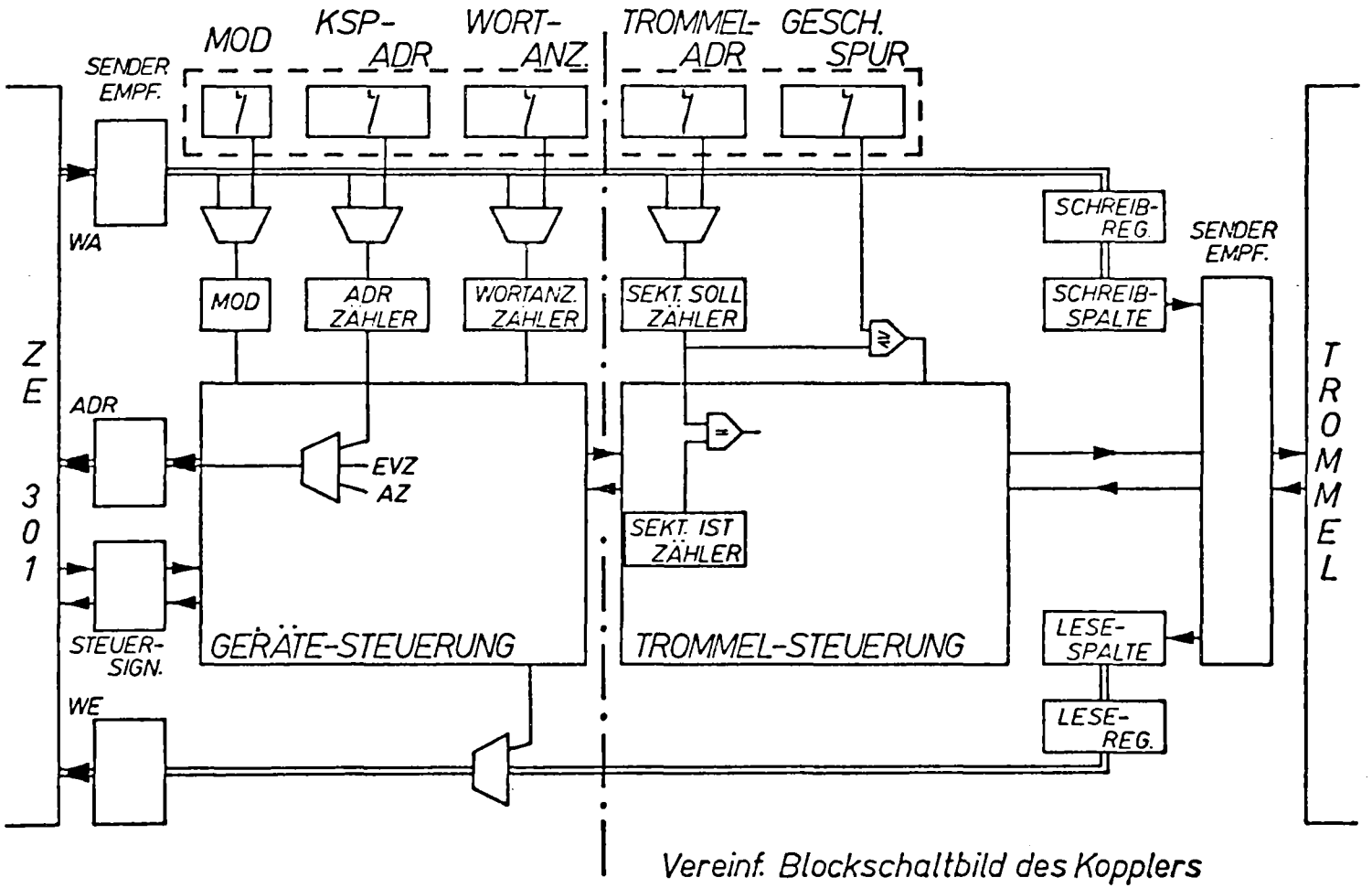
Die Arbeitsspeichernahtstelle der S 301 besteht bekanntlich hardwaremäßig aus den Dateneln- und Ausgabeleitungen (je 24), den 14 Adressleitungen sowie verschiedener Steuerleitungen. Die Parameterversorgung erfolgt in der Weise, daß in einer festen Arbeitsspeicherzelle, der Elementversorgungszelle EVZ, nacheinander die Parameter Arbeitsspeicheranfangsadresse, Blocklänge sowie Trommeladresse abgelegt werden, wobei der Koppler jeweils durch ein Steuersignal BBE (Befehlt steht bereit), durch den Elementversorgungsbefehl EV ausgelöst, veranlaßt wird, den Parameter zu lesen. Nach Übernahme des letzten Parameters wird der Blocktransfer automatisch gestartet. Soll ein Block auf die Trommel geschrieben werden, so wird nach der Blockanfangskennung ein Wort aus dem Arbeitsspeicher gelesen und nach Zwischenspeicherung im Schreibregister über die Schreibspalte seriell auf die Trommel geschoben. Nach jedem Wort wird der Adresszähler erhöht, der Wortanzahlzähler vermindert, bis der gesamte Block übertragen ist. Danach wird mit dem Paritywort sowie einer Blockendekennung der Block abgeschlossen. Nach Übertragung der Fehlerzustandskennung in die Anzeigenstelle AZ des Arbeitsspeichers wird das Ende des Blocktransfers der Zentraleinheit mit einer BAP gemeldet.

Bei dem von uns gebauten Koppler handelt es sich nicht um einen Nachbau sondern um eine völlige Neuentwicklung, was sich z.B. daran zeigt, daß der Siemenskoppler weitgehend synchron mit einem Taktraster, dieser jedoch asynchron arbeitet, was sicher nicht von Vorteil ist; das Gesamtprojekt muß jedoch unter den Ausbildungsgesichtspunkten gesehen werden.
Gegenüber dem Standardkoppler wurden drei Erweiterungen vorgenommen.
1. Beim Laden des Organisationsprogramms wird das Vorhandensein eines Externspeichers daran erkannt, daß nach einem EV-Befehl die LVZ gelöscht ist. Unterdrückt man im Koppler beim Lesen der LVZ das Löschen, so wird die Trommel nicht erkannt, obwohl man auf der Basis der Maschinenbefehle mit ihr arbeiten kann.
Dies war während der Inbetriebnahmephase von Interesse, um den normalen Betrieb nicht zu stören. Durch einen Schalter kann zwischen den beiden Betriebsarten umgeschaltet werden.
2. Von größerer Bedeutung ist die Möglichkeit, Blocktransfers auch manuell durchführen zu können. So kann z.B. per Knopfdruck ein Urlader von der Trommel in den Kernspeicher geladen werden, oder aber es kann, wie bei uns z.Zt. üblich, eine Standardversion des ORG einschließlich der üblicherweise benötigten Hilfsprogramme, wie PROTEST usw., durch einen Knopfdruck von der Trommel in den Kernspeicher einkopiert werden. Da bei unserer Ausbildung häufig von den Studenten auf unterster Ebene gearbeitet wird und hierbei das Organisationsprogramm fast regelmäßig zerstört wird, ist diese Möglichkeit von besonderer Bedeutung. Hierzu können an einem Schalterfeld die Parameter Kernspeicheradresse, Wortanzahl und Trommeladresse sowie die Transferrichtung(MODUS) eingestellt werden. Durch einen Tastendruck werden die entsprechenden Register vom Schalterfeld geladen sowie der Transfer gestartet.
3. Damit wichtige Programme nicht auch auf der Trommel zerstört werden können, ist ein Hardwareschreibschutz vorgesehen. Ist die Trommeladresse, je nach Einstellung größer oder kleiner als der auf dem Schalterfeld eingestellte Wert , so wird der Schreibvorgang abgebrochen und eine entsprechende Fehlermeldung an die Zentraleinheit gegeben.

Der Aufbau erfolgte in einem 19"-Schrank. Wie die Abbildung zeigt, befinden sich in diesem Schrank von unten nach oben:
Lüfter, Heliumversorgung, Trommelspeicher mit Schreib-Lese-Elektronik, Schalterfeld, Koppler und Stromversorgung.

Bei dem Koppler erkennt man links die Anschlüsse der Arbeitsspeichernahtstelle, rechts die Anschlüsse an die Trommel.
Der Aufbau des Kopplers erfolgte in Wire-Wrapp-Verdrahtung auf Europakarten in Standard-TTL-Technik.
Diese Aufbautechnik stellt nach unserer Erfahrung für Prototyp-Aufbauten die beste Art dar. Die Schaltungen lassen sich relativ einfach erstellen, sind zuverlässig und übersichtlich und lassen sich einfach testen, korrigieren und erweitern.

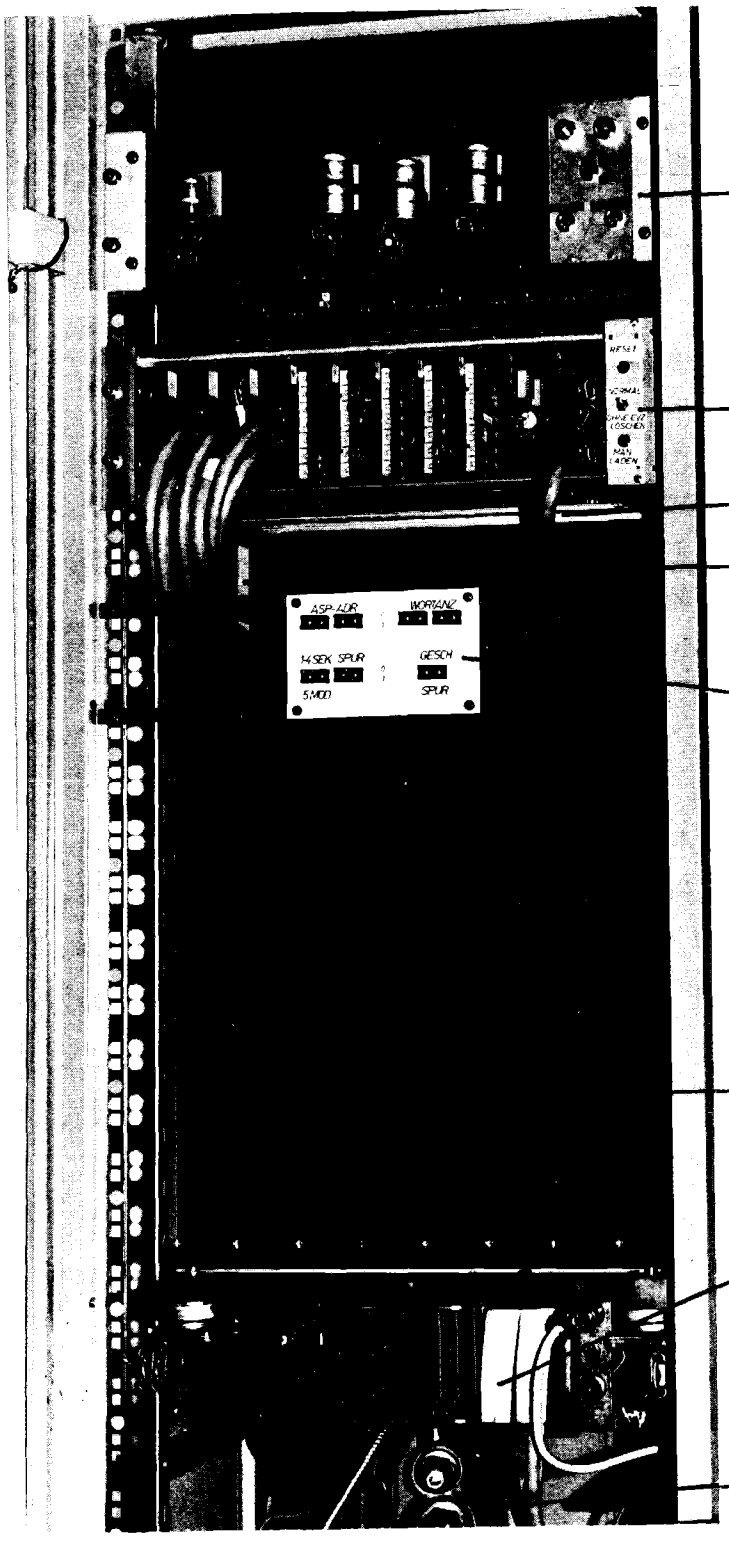
Durch diese Arbeit steht uns inzwischen eine Proezßrechneranlage S 301 zur Verfügung, die insbesondere für unsere Ausbildungszwecke wesentlich an Effektivität gewonnen hat.
Durch die Arbeit selbst konnten die hieran beteiligten Studenten viele Erfahrungen gewinnen, die ihnen in ihrem weiteren beruflichen Werdegang sicher von Nutzen sein werden. Unser Dank gilt neben diesen Studenten auch der Firma Siemens für ihre freundliche Unterstützung.



Vereinf. Blockschaltbild des Kopplers
(GH Paderb., Abt. Soest)

Bild 1

Bild 2



Stromversorgung

Koppler

zur Trommel

zur ZE 301

Schalterfeld

Trommel

mit Lese-Schreibelektronik

Heliumversorgung

Lüfter

Trommelspeicher zur ZE S301, GH PADERBORN/FB 16/SOEST

Besichtigungen

Während der Jahrestagung wurde die Möglichkeit zu folgenden Besichtigungen gegeben:

1. Badenwerk
   Einsatz einer S330 in der Netzleitstelle
   Daxlanden

2. Universität Karlsruhe
   Institut für Informatik III
   Lehrstuhl: Prof. Dr. G. Krüger
   Praktikumsversuch: Regelprozeß mit S320
   PRINT und BASIC auf S330

3. Universität Karlsruhe
   Institut für Informatik III
   Lehrstuhl: Prof. Dr. U. Rembold
   Waschmaschinen-Prüfstand mit S310
   Steuerung einer Widerstandsschleifmaschine mit S310
   Mustererkennung

4. Universität Karlsruhe
   Rechenzentrum
   Lehrstuhl: Prof. Dr. A. Schreiner
   Rechnernetz S310/S330/UNIVAC/B7700

5. Kernforschungszentrum Karlsruhe
   Institut für Datenverarbeitung in der Technik II
   Leitung: Prof. Dr. H. Trauboth
   Datenbanksystem FADABS auf S330
   Testsystem RXVP auf S330

SAK JAHRESTAGUNG KARLSRUHE 1978 TEILNEHMERVERZEICHNIS
07.04.1978

1	ABEND, PETER	HMI BERLIN
2	ADAMS, HERBERT	SIEMENS AACHEN
3	ANKERMANN, HEINZ	SIEMENS DORTMUND
4	BACHNER, ERHARD	KFA JUELICH
5	BAMBERGER, K.F.	SIEMENS KARLSRUHE
6	BECKER, DR. WILHELM	GHS PADERBORN
7	BENZINGER, FRANZ	SIEMENS MUENCHEN
8	BERNARDS, PETER	SIEMENS KARLSRUHE
9	BISCHOF, DIETHART	SIEMENS NUERNBERG
10	BLISCHKE, HELGE	SRZ BERLIN
11	BORRMANN, HARTMUT	KFK/IDT KARLSRUHE
12	BRAUN, PETER	SIEMENS KASSEL
13	BRUEGNER	PORSCHE WEISSACH
14	BUSECK, HERMANN	BA FUER STRASSENWESEN KOELN
15	DE JONG, E.	ESA/ESOC DARMSTADT
16	DIEMER, JOCHEN	SIEMENS STUTTGART
17	DRAWERT, K.-H.	SIEMENS KARLSRUHE
18	ECKERT, DIETRICH	UNIVERSITAET KARLSRUHE
19	ERNST, ROLAND	GHS KASSEL
20	FALKENBERG, ERNST-ALBERT	SIEMENS KARLSRUHE
21	FISCHER, SIEGFRIED	PTB BRAUNSCHWEIG
22	FRIEHMELT, DR. RUEDIGER	KFK/IDT KARLSRUHE
23	GAIS, PETER	GSF NEUHERBERG
24	GEIGER, WERNER	KFK/IDT KARLSRUHE
25	GMEINER, LOTHAR	KFK/IDT KARLSRUHE

SAK JAHRESTAGUNG KARLSRUHE 1978 TEILNEHMERVERZEICHNIS
07.04.1978

26	GRUESSINGER, G.	UNIVERSITAET KARLSRUHE
27	GUHR, WOLFRAM	UNIVERSITAET DORTMUND
28	HAAF, WILFRIED	BERUFSFOERDERUNGSWERK HEIDELBERG
29	HAELSIG, MICHAEL	KFK/IDT KARLSRUHE
30	HAUPTMANN, GERHARD	BERUFSFOERDERUNGSWERK HEIDELBERG
31	HEIM, KLAUS	UNIVERSITAET KARLSRUHE
32	HEINING	RWE PULHEIM
33	HESPELT, FRITZ	SIEMENS MUENCHEN
34	HIRSCHBERG, GUENTER	SIEMENS ERLANGEN
35	HOCHMUTH, HEINZ	DORNIER FRIEDRICHSHAFEN
36	HOCHREIN, EDGAR	FICHTEL&SACHS SCHWEINFURT
37	HOEGER, HANS	SIEMENS KARLSRUHE
38	HOLLER, DR. ELMAR	KFK/IDT KARLSRUHE
39	HOMMEL, DR. GUENTER	KFK/PDV KARLSRUHE
40	HUBER, HERMANN	SIEMENS KARLSRUHE
41	HUPPERTZ, H.	KFA JUELICH
42	HUSHEER, FRANK	SIEMENS KARLSRUHE
43	JABS, E.	ESA/ESOC DARMSTADT
44	JAESCHKE, DR. ANDREAS	KFK/IDT KARLSRUHE
45	JORDAAN, JOHANNES	GHS SIEGEN
46	JUENEMANN, HANS-PETER	SIEMENS HAMBURG
47	JUST, GERHARD	SIEMENS KARLSRUHE
48	KAELBER, GERLINDE	SIEMENS MUENCHEN
49	KAMMLER, ERWIN	RWE PULHEIM
50	KERPE, ROLF	KFK/IDT KARLSRUHE

SAK JAHRESTAGUNG KARLSRUHE 1978 TEILNEHMERVERZEICHNIS
07.04.1978

51	KEVEKORDES	GHS PADERBORN
52	KIESSLING, DR. REINER	ACCU-FABRIK HAGEN SOEST
53	KINDT, THOMAS	GHS KASSEL
54	KLEIN, ROBERT	FGAN MECKENHEIM
55	KLOECKER, ADOLF	KFA JUELICH
56	KOCH, G.	UNIVERSITAET KARLSRUHE
57	KOHLENBECK, KARL-HEINZ	GHS SIEGEN
58	KOKEMOOR, WILFRIED	HOESCH DORTMUND
59	KOSSMANN, PETER	KFK/IDT KARLSRUHE
60	KRISTEN, W.	TH AACHEN
61	KUHN, JUERGEN	SIEMENS KARLSRUHE
62	LANDMARK, KLAUS	KFK/IDT KARLSRUHE
63	LAUQUE, J.P.	ESA/ESOC DARMSTADT
64	LEMPERLE, WALTER	KFK/IDT KARLSRUHE
65	LIEDTKE, HEINZ	SIEMENS KARLSRUHE
66	LOUP, LOTHAR	KFA JUELICH
67	MACKE, ERNST	FGAN MECKENHEIM
68	MANGOLD, EBERHARD	SIEMENS STUTTGART
69	MOSKO, CHRISTOPHORUS	SESA DUESSELDORF
70	MUELLER, GOTTFRIED	SIEMENS KARLSRUHE
71	MUSSGNUG, GERNTIFL	UNIVERSITAET KARLSRUHE
72	NAUNDORF, DR. FRIEDRICH	SUEDZUCKER MANNHEIM
73	NEIDMANN, TILO	DAIMLER BENZ STUTTGART
74	NEUBERT, DR. WALTER	PTB BRAUNSCHWEIG
75	NOACK, GUENTER	DFVLR HARDTHAUSEN

SAK JAHRESTAGUNG KARLSRUHE 1978 TEILNEHMERVERZEICHNIS
07.04.1978

76	OHLMER, EVA	SIEMENS MUENCHEN
77	PETERS, HARTMUT	KFA JUELICH
78	PIEPER, K.	SIEMENS KARLSRUHE
79	PIEPER, PROF. F.	FH ULM
80	POHL, E.	TU BRAUNSCHWEIG
81	POLSTER, FRANZ J.	KFK/IDT KARLSRUHE
82	RAMOLLA, DR. BERNHARD	HOECHST BOBINGEN
83	REINHART, P.	KFA JUELICH
84	RINDER, JOSEF	SIEMENS KARLSRUHE
85	RONGEN, F.	KFA JUELICH
86	ROSSIRE, PIERRE	SBB BERN
87	ROTHERMEL, HANS	SIEMENS KARLSRUHE
88	RUEHLING, HORST	HOESCH DORTMUND
89	SANDER, ERNST	HOESCH DORTMUND
90	SCHLEER, HORST	HOECHST BOBINGEN
91	SCHLESIGER, HORST	SIEMENS KASSEL
92	SCHMIDT, HERBERT	UNIVERSITAET BONN
93	SCHMIDT, PETER	BERUFSFOERDERUNGSWERK HEIDELBERG
94	SCHNEIDER, K.	SIEMENS KARLSRUHE
95	SCHOKNECHT, HARALD	SIEMENS KARLSRUHE
96	SCHROEDTER, H.D.	HMI BERLIN
97	SCHROTH, PETER	UNIVERSITAET KARLSRUHE
98	SCHULTESS, GERT	RBU HANAU
99	SCHUSTER, DR. H.-J.	PTB BRAUNSCHWEIG
100	SCHWAN, U.	TH AACHEN

SAK JAHRESTAGUNG KARLSRUHE 1978 TEILNEHMERVERZEICHNIS
07.04.1978

101	SCHWEITZER, HELMUTH	DAIMLER BENZ STUTTGART
102	SETHMACHER, HILMAR	UNIVERSITAET KARLSRUHE
103	STEGMAIER, RAINER	DAIMLER BENZ STUTTGART
104	STEHLING, K.	ESA/ESOC DARMSTADT
105	SUTER, DR. EDGAR	SBB BERN
106	TENTEN, W.	KFA JUELICH
107	THIEL, AXEL	PTB BRAUNSCHWEIG
108	TRAUBOTH, PROF. DR. HEINZ	KFK/IDT KARLSRUHE
109	TRETTER, GERD	KFK/EKS KARLSRUHE
110	UHLMANN, BERND	HMI BERLIN
111	URBANETZ, MANFRED	FICHTEL&SACHS SCHWEINFURT
112	VOGES, UDO	KFK/IDT KARLSRUHE
113	VOSS, PROF. MARTIN	FH FLENSBURG
114	WEISE, PROF. DR. KLAUS	PTB BRAUNSCHWEIG
115	WENZEL, THEO	SIEMENS KARLSRUHE
116	WOLFINGER, BERND	KFK/IDT KARLSRUHE
117	WOLFRAM, EVELYN	HMI BERLIN
118	ZOELLER, GERNOT	SIEMENS KARLSRUHE

Protokoll
der SAK-Mitglieder-Versammlung am 6.4.78

TOP 1: Bericht des Vorstandes.

Der Vorstand (geschäftsführender Ausschuß) traf sich am 7.1o.77 in Karlsruhe und besprach folgende Themen:

1. SAK-Tagung 78,
2. Erhöhung des Bekanntheitsgrades des SAK über die ZNs der Siemens AG,
3. Aktualisierung der Mitglieder- und Installationen-Kartei,
4. Programm-Austausch(Versendung von Fragebogen, s.u.)

TOP 2: Satzungsänderungen

Die Mitglieder-Versammlung beschloß einige Änderungen der SAK-Satzung:

1. Der Beschluß der Mitglieder-Versammlung vom 21.4.77, die SAK-Mitteilungen nur noch bei Bedarf herauszugeben, mußte in der Satzung noch berücksichtigt werden.
2. Der Tagungsbeauftragte wird künftig von der Institution benannt, die - gewählt von der Mitgliederversammlung - die Jahrestagung veranstaltet.
3. Tagungsbeauftragter und Geschäftsführender Ausschuß erhalten die Möglichkeit der Auswahl von Tagungs-Vorträgen, wofür die Satzung einen möglichst weit gesteckten Rahmen vorsehen muß:
   Um Gastvorträge zu ermöglichen, sind Themen allgemeiner Art zugelassen.
   Um unerwünschte Werbung verhindern zu können, ist die Themenauswahl für Spezialvorträge eingeschränkt.

TOP 3: Mitglieder-Kartei

Das SAK-Sekretariat wird die Mitglieder-Kartei auf den neuesten Stand bringen und dazu in Kürze über alle ZNs Fragebogen verschicken, um alle Installationen genau zu erfassen.

TOP 4: Programm-Vorhaben

Zur SAK-Tagung 78 waren mit den Einladungen Fragebogen verschickt worden, die bestehende und geplante Software

zum Zwecke des Programm-Austausches erfassen sollten. Diese Aktion war vom geschäftsführenden Ausschuß auf seiner Sitzung am 7.1o.77 beschlossen worden. Bis zum Beginn der SAK-Tagung 78 sind 28 Fragebogen zurückgekommen, sodaß es mit der zugrundeliegenden Information erstmals sinnvoll erscheint, eine 'Software-Börse' des SAK einzurichten. Die Ergebnisse der Auswertung sollen in den nächsten SAK-Mitteilungen veröffentlicht werden.

TOP 5: Wahlen

Der amtierende geschäftsführende Ausschuß beendete seine Amtszeit. Der bisherige Vorsitzende kandidierte nicht wieder für den neu zu wählenden Ausschuß, da an der Fachhochschule Ulm eine Rechner-Ausschreibung laufe.
Für die Amtszeit von einem Jahr (bis zur SAK-Tagung 79) wurde - bereits nach der neuen Satzung - folgender Geschäftsführender Ausschuß gewählt:

Herr Abend, Hahn-Meitner-Institut Berlin
Herr Bachner, Kernforschungsanlage Jülich
Herr Pohl, Technische Universität Braunschweig
Herr Voges, Kernforschungszentrum Karlsruhe

Als Tagungsort für die 1o. SAK-Jahrestagung 1979 wurde mit großer Mehrheit das Hahn-Meitner-Institut Berlin gewählt. Für den Fall dieser Wahl war Herr Abend vom HMI bereits als Tagungsbeauftragter benannt worden. Daraus ergibt sich, daß der Geschäftsführende Ausschuß für 78/79 nur aus 4 Mitgliedern besteht.

Nachtrag:
Bei seiner konstituierenden Sitzung am 6.4.78 abends wählte der Geschäftsführende Ausschuß Herrn Abend (nicht nur der abendlichen Stunde wegen) wiederum zum SAK-Vorsitzenden.

F. Pieper

Adressen des SAK-Vorstands

Dipl.-Ing. Peter Abend
Hahn-Meitner-Institut für Kernforschung
Sektor Elektronik
Glienicker Straße 100
1000 Berlin 39
Tel.: 030/8008-520

Dipl.-Phys. Erhard Bachner
Kernforschungsanlage Jülich - ZAT
Postfach 1913
5170 Jülich
Tel.: 02461/613284

Dipl.-Ing. E. Eberhard Pohl
Technische Universität Braunschweig
Institut für elektrische Maschinen,
Antriebe und Bahnen
Postfach 3329
3300 Braunschweig
Tel.: 0531/391-3910

Dipl.-Math. Udo Voges
Kernforschungszentrum Karlsruhe
Institut für Datenverarbeitung in
der Technik - IDT -
Postfach 3640
7500 Karlsruhe 1
Tel.: 07247/82-3925

SAK-Sekretariat:
Dipl.-Ing. Peter Bernards
Siemens AG - E 612 -
Rheinbrückenstraße 50
7500 Karlsruhe 21
Tel.: 0721/595-2351