W. Ehrenberger
Technische Hochschule
München

"Alarmbearbeitung in einer 3o5 und Simultanarbeit in höheren Sprachen"

Alarmbearbeitung in einer 305 und Simultanarbeit in höheren Sprachen

W. Ehrenberger 12. 8. 70

Zusammenfassung:

Die folgenden Ausführungen beziehen sich auf die Hybridanlage am Institut für Meß- und Regeltechnik an der TH-München. Für manche Hybrid-Anwendungen ist der über das Org 300 abgewickelte Externverkehr zu langsam. Eine zusätzlich eingebaute Alarmbearbeitung ermöglicht eine rasche Reaktion auf Anrufe an die Zentraleinheit. Um aus Alarmprogrammen Externverkehr abwickeln zu können, muß das Org etwas verändert werden. Die Prozeßelemente P3AS und P3ES/N sind mit einem Zusatz auszurüsten. In höheren Sprachen ist zur Zeit keine Simultanarbeit zwischen einem Programm und dessen eigener Ein-Ausgabe möglich. Codeprozeduren können eine solche Simultanarbeit gestatten, indem sie den Ein-Ausgabeaufruf von seinem EXWA trennen und in einem Vorlauf Org-Makros absetzen.

1. Alarmbearbeitung

1.1 Problemstellung

Nehmen wir an, wir wollen für den Analogrechner einen z-Wert zu einem vorgegebenen Paar von x-y-Werten bestimmen.

Bild 1a, b

Wir müssen folgendes tun:

1. Ein Wertepaar xy einlesen
2. den z-Wert durch Interpolation ermitteln
3. den z-Wert ausgeben.

Dies soll sehr schnell vor sich gehen, um den Zeitfehler klein zu halten.

Der Vorgang soll auch unabhängig von einem anderen Hybridprogramm in der 305 ablaufen können.

Wie man leicht einsieht, ist die durch das Org 300 geleistete Organisationsarbeit dazu zu zeitraubend. Ebenso ist die selbst organisierte BAP-Bearbeitung zu langsam, weil immer auf eine BAP-unterbrechbare Stelle gewartet werden muß und weil das Org nicht BAP unterbrechbar ist.

1.2 Einbau einer Alarmunterbrechungsmöglichkeit in eine 305

Wir haben deshalb das Steuerwerk etwas erweitert, um ein rasches Reagieren der Zentraleinheit zu ermöglichen. Diese Alarmbearbeitung hat folgende Eigenschaften:

1. Sie erlaubt eine Programmunterbrechung nach jedem Befehl, der kein Elementbefehl ist.

2. Sie setzt das Programm an einer durch Schalter einstellbaren Adresse fort.

3. Sie stellt nicht in Zelle 0 sicher, sondern in einer ebenfalls einstellbaren Zelle.

4. Sicher gestellt wird nicht nur der Befehlszähler, sondern auch das Elementauswahlregister, die Überlaufbits bei der Akkus und beide Bits 0 der Akkus (Bild 2)

5. Die Priorität dieser Unterbrechung liegt zwischen der MAP u.d. BAP.

6. Die MAP bleibt in ihrer bisherigen Funktion voll erhalten.

7. Die Alarmbearbeitung verhindert eine BAP solange irgendein Alarm ansteht.

Um diese Punkte zu erfüllen, mußte das Steuerwerk mit Zubauten versehen werden (Bild 3).

2 Platten und 1 Stecker waren neu einzufügen.
An 5 Einbauplätzen wurden Veränderungen nötig.

Im Rechenwerk wurden 2 Einbauplätze neu belegt.

1.3 Reaktionsgeschwindigkeit der Hardware

Die Reaktiongeschwindigkeit wurde durch den Einbau bedeutend erhöht. Die maximale Reaktionszeit der Hardware liegt bei der Dauer eines Befehls, also unter 40µs. Ist der Befehl MAP-unterbrechbar gewesen, kommt u.U. noch eine PUN-Laufzeit = 4,5µs dazu. Bei kurzen Befehlen, dringt der Alarm bereits nach 1,5µs durch.

Selbstverständlich ist aubh das Org unterbrechbar.

1.4 Beginn und Ende von Alarmprogrammen

Nachdem die Unterbrechung hardwaremäßig stattgefunden hat, müssen noch die Registerinhalte sichergestellt werden. Man hat also die Befehle zu durchlaufen:

```
        SPR   (    )      Einsprung über die Sprungleiste
ALPROGR TAS'  SI          Sicherstellung des RA
        TEP'  1183        Holen der Rücksprunginformation
        EA5   ....        Zulassen von Alarmen höherer Priorität
        TAS'  SI+1        Sicherstellen der Rücksprunginf.
        TAS   SI+2              "        des LA
        TAX   SI+3              "        des ER
```

Mit dem EA5 kann man Unterbrechungen höherer Priorität zulassen.

Am Ende der dann zu rechnenden Unterbrechungsroutine müssen aus dem Bereich SIZE die Registerinhalte wieder restauriert werden, die Bits 0, die Überlaufsbits und das Elementauswahlregister in den alten Zustand gebracht werden; dann erst kann man zurückspringen an die Stelle, an der man unterbrochen hatte.

Der Schluß eines Alarmprogramms hat die Befehle:

```
        EA3   ....        Abtaster frei-geben, auch Alarme
                          niedrigerer Priorität werden möglich
        EPR   (    )      Restaurieren des Elementauswahlregisters
        SPR   (SI+1)      Rücksprung in das unterbrochene
        SPR   (SI+1)      Programm
```

Nachdem hinter Elementbefehlen keine Unterbrechung durch Alarm möglich ist, ist diese Befehlsfolge vom Wiederzulassen von Alarmen bis zum Rücksprung nicht unterbrechbar.

Die gesamte Zeit für das Wiederherstellen der Registerinhalte usw am Alarmende beträgt 69µs.

1.5 Externverkehr aus Alarmprogrammen

Aus einer in der vorhin beschriebenen Weise eingeleiteten Alarmroutine kann man natürlich keinen Externverkehr machen zu Elementen, die über das Org angesprochen werden. Das Org ist ja u.U. selbst unterbrochen worden! Wenn man das Element über EA-, EV-Befehle versorgt, so gerät man in die Gefahr ein bereits versorgtes Element neu zu versorgen und die alte Versorgung zu zerstören, löst aber immer eine Übergabe von PU-Bits und BAP aus, was dann wieder zu einer Org-Bearbeitung führt, Zeit kostet und im allgemeinen auf eine UAP führt.

Man muß also auch im Org selbst noch etwas tun, nämlich verhindern, daß die eigentliche Versorgung des Elements unterbrochen werden kann. Dies erreicht man durch Hintereinandersetzen der EA-EV-Befehle; denn Elementbefehle haben wir ja ununterbrechbar gemacht.

Im Element selbst müssen wir auch etwas verändern. Wir müssen auf Anfrage das Signal "Abweisend" geben können und wir müssen dieses Signal nach Ende des jeweiligen Puffers wieder zurücknehmen. Wir müssen ferner in der Lage sein, die BAP zu verhindern. Diese Forderungen füllt ein Zusatzeinbau mit den Eigenschaften:

1. Ein EA1 bringt ABW, wenn das Element versorgt ist.
2. Ein EA2 verhindert die BAP am Pufferende.
3. ABW verschwindet, sobald beide Puffer abgelaufen sind.

Vor jeder Elementversorgung muß mit einem EPR festgestellt werden, ob das Element zugänglich ist. Dies wurde ebenfalls in den die P3-Elemente versorgenden Org-Teil aufgenommen.

Im Alarmprogramm muß der auf Prosa-Ebene programmierende Anwender selbst auf „Abweisend" prüfen. Mit einem EA1-Befehl kann er das Abweisend hervorrufen, falls eine Org-Versorgung vorher stattgefunden hatte, Der Anwender kann

nach Wunsch aber auch eine bereits gegebene Versorgung überschreiben.

1.6 Alarmbearbeitung in höheren Sprachen

Wenn man nun in einer höheren Sprache mit Alarmen arbeiten möchte, dann muß man für jeden der verschiedenen Alarme je 1 Codeprozedur vorsehen, die den Alarmbeginn und das Alarmende ausführt. Für alle Alarme nur 1 oder 2 Codeprozeduren zu verwenden ist bei einer 305 nur auf Kosten der Rechenzeit oder größeren Hardwareaufwands möglich. In einem Vorlauf muß die richtige Einsprungadresse in das Alarmprogramm eingetragen werden. Alarmprogramme in einer höheren Sprache sind als Subroutinen der Coderoutinen möglich. Bei Verwendung von Standardfunktionen ist Vorsicht geboten. Ein-Ausgaben der üblichen Art sind nicht erlaubt. Bild 4.

Nachdem jede BAP-Bearbeitung per Hardware verhindert wird, solange irgend ein Alarm ansteht, entstehen keine Schwierigkeiten durch die BAP-unterbrechbaren Stellen eines durch einen Compiler erzeugten Programmstücks.

2. Simultanarbeit in höheren Sprachen

2.1 Problem

Bei der Digitalisierung von Analogbändern tritt folgende Aufgabenstellung auf:

Während ein Puffer über ein P3-Element gefüllt wird, muß der andere Puffer versorgt und weggeschaft werden. Der Datenfluß vom Analogrechner zum Digitalrechner muß kontinuierlich sein; er darf nicht durch Organisationsrechenzeiten unterbrochen werden. Bild 5.
Diese Aufgabe ist in Prosa leicht lösbar.

In einer höheren Sprache aber bekommt man Schwierigkeiten. Standardfunktionen in höheren Sprachen pflegen gleich hinter den Ein-Ausgabe-Aufruf den Warteaufruf zu enthalten. Es ist keine Simultanarbeit zwischen Externverkehr und dem betreffenden Anwenderprogramm möglich. Probleme von der Art des aufgeworfenen können nur durch mehrere simultan arbeitende Programme in einer anwenderorientierten Sprache gelöst werden.

In dem angesprochen Fall könnte man zudem wünschen, sich im Digitalrechner zur Zwischenpufferung eines Ringspeichers zu bedienen. Ein solcher Speicher, der aus mehreren Einzelpuffern besteht und von einer Seite ständig gefüllt und von der anderen geleert wird, ist in einer höheren Sprache ebenfalls nicht programmierbar.

Schließlich möchte man je nach Schärfe der Geschwindigkeitsforderungen die Möglichkeit haben, die eingelesenen Daten sofort umzucodieren oder sie erst nach einer Abspeicherung auf einem Massenspeicher zur Verarbeitung aufzubereiten.

2.2 Forderungen an die höhere Sprache

Aus dem vorhergehenden Punkt lassen sich für eine anwenderorientierte Sprache für einen Hybridrechner die Wünsche angeben:

1. Die Ein-Ausgabeaufrufe müssen von den Warteaufrufen getrennt werden.

2. Warteschlangenbildung ist zuzulassen. Dazu müssen so viele Org-Makros geschaffen werden können, wie eine eventuelle Warteschlange innerhalb eines Programms Glieder haben soll.

3. Die Umcodierung soll von der Ein/Ausgabe unabhängig sein.

Zur Erreichung dieser Ziele wurden Codeprozeduren erstellt.

2.3 Schema der Codeprozeduren

Die Ein/Ausgabeaufrufe zur Simultanarbeit müssen sich zunächst einmal die benötigten Org-Makros schaffen.

Dazu bedient man sich eines Vorlaufs. In diesem Vorlauf trägt die Codeprozedur den gewünschten Org-Makro zusammen mit seinem Exwa in einem vorher zu deklarierenden Feld ein. Nach der Zahl dieser Felder und Eintragungen bestimmt sich die Maximallänge einer möglichen Warteschlange.

Im Rechenlauf wird der abgesetzte Aufruf ausgeführt. Dabei gibt es zwei Möglichkeiten:

1. Die Parameter, die das Externe Element betreffen, werden verändert.

2. Die das EXE betreffenden Parameter bleiben unverändert.

Ein Wartelauf schließlich erlaubt das Geben des Exwas. Bild 6.

2.4 Ein Beispiel

Nehmen wir die Codeprozedur TYDT, Type Data.
Sie wird deklariert:

```
TYDT(NR, ADR, DIST, PLA).,
'VALUE' NR, DIST.,
'INTEGER' NR, DIST.,
'INTEGER' 'ARRAY' ADR, PLA.,
'CODE'.,
```

ADR bezeichnet das bereits umcodierte Feld aus dem ab der Entfernung DIST vom Beginn ausgegeben werden soll. PLA muß auch deklariert worden sein als:

```
'INTEGER' 'ARRAY' PLA (/ 1.. 10/).,
```

Ist NR = 0, so wird in PLA eingetragen:

```
PLA     UNT  (5)
        003  19
        NOP  0
        NOP  ADR + DIST
        NOP   0
        SPR  (RUECK)
        UNT  (5)
        001  17
        NOP  PLA
        SPR  (RUECK)
```

Ist NR = 2, wird auf PLA (/1/) gesprungen. Ist NR = 3 oder größer, wird auf PLA (/7/) gesprungen. Ist NR = 1, wird entsprechend der neuen aktuellen Parameter ADR u. DIST die entsprechende Eintragung in PLA (/4/) korrigiert und anschließend auf PLA (/1/) gesprungen.

2.5 Vorhandene Aufrufe

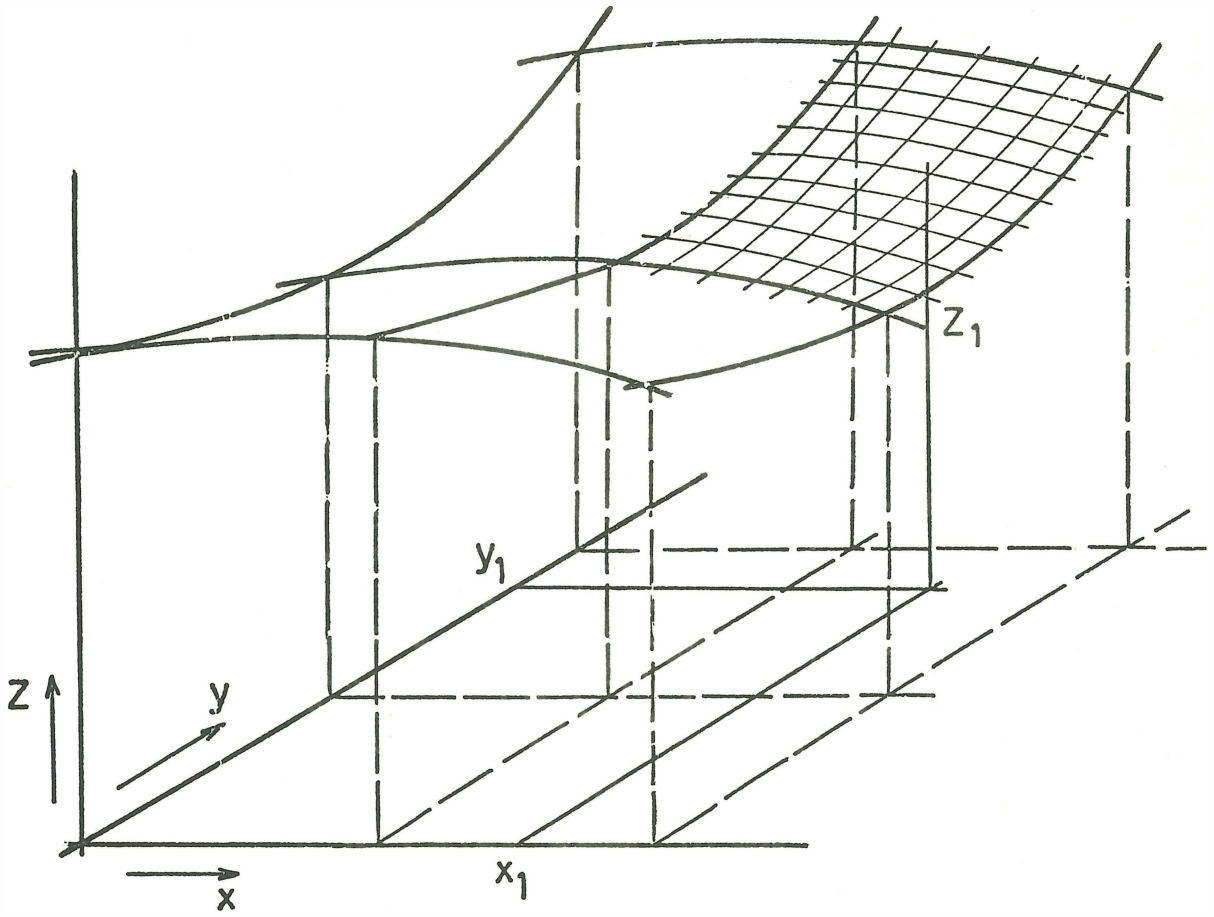
In der eben geschilderten Weise wurden folgende Aufrufe für Algol zugänglich gemacht.

```
RDCD (NR, ADR, DIST, N, PLA),,
Read cards, Lochkarteneingabe;

TYDT (NR, ADR,,DIST, PLA).,
Type Data , Blattschreiberausgabe  von Daten;

TYTX (NR, TEXT, PLA).,
Type Text , Blattschreiberausgabe von Text;

PRIN (NR,ADR, DIST, LINE, PLA).,
Print , Schnelldruckerausgabe
```

TAPE (NR, MODE, TNR, ADR, DIST, N, PLA).,
Magnetband Ein-Ausgabe, Bandsteuerung;

COPL (NR, MODE, ADR, N, PLA).,
Coppiung zweier Digitalrechner;

DAAD (NR, BLOCK, DISTBL, NBLOCK, ADR, DIST, N, REG, PLA).,
Digital-Analog-Analog-Digital, simultane Ein-Ausgabe zum Analogrechner.



Zu einem Wertepaar (x,y) wird der
Z-Wert errechnet

Bild 1a

Räumliche Darstellung der Interpolation eines Wertes z_1 anhand eines vorgegebenen Wertepaars x_1,y_1, wenn ein festes Raster von Werten x,y,z gespeichert ist

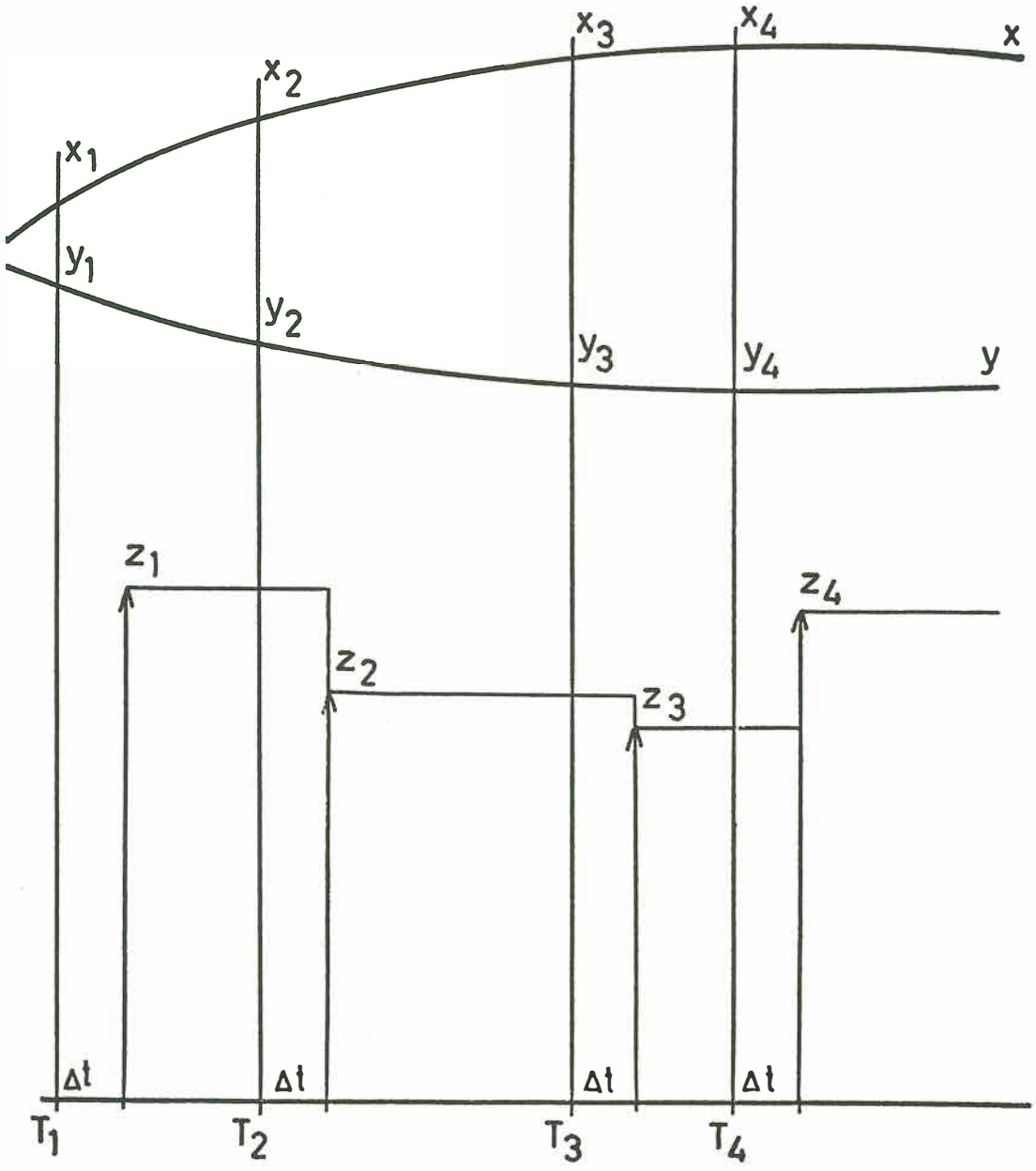


Bild 1b

Zeitliche Darstellung der Interpolation von z-Werten aufgrund einzulesender x,y-Werte

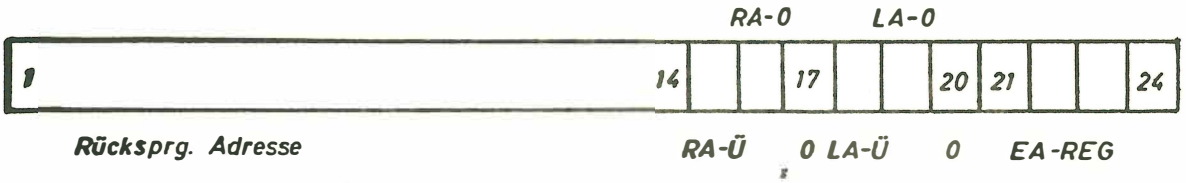


Bild 2
Rücksprunginformation

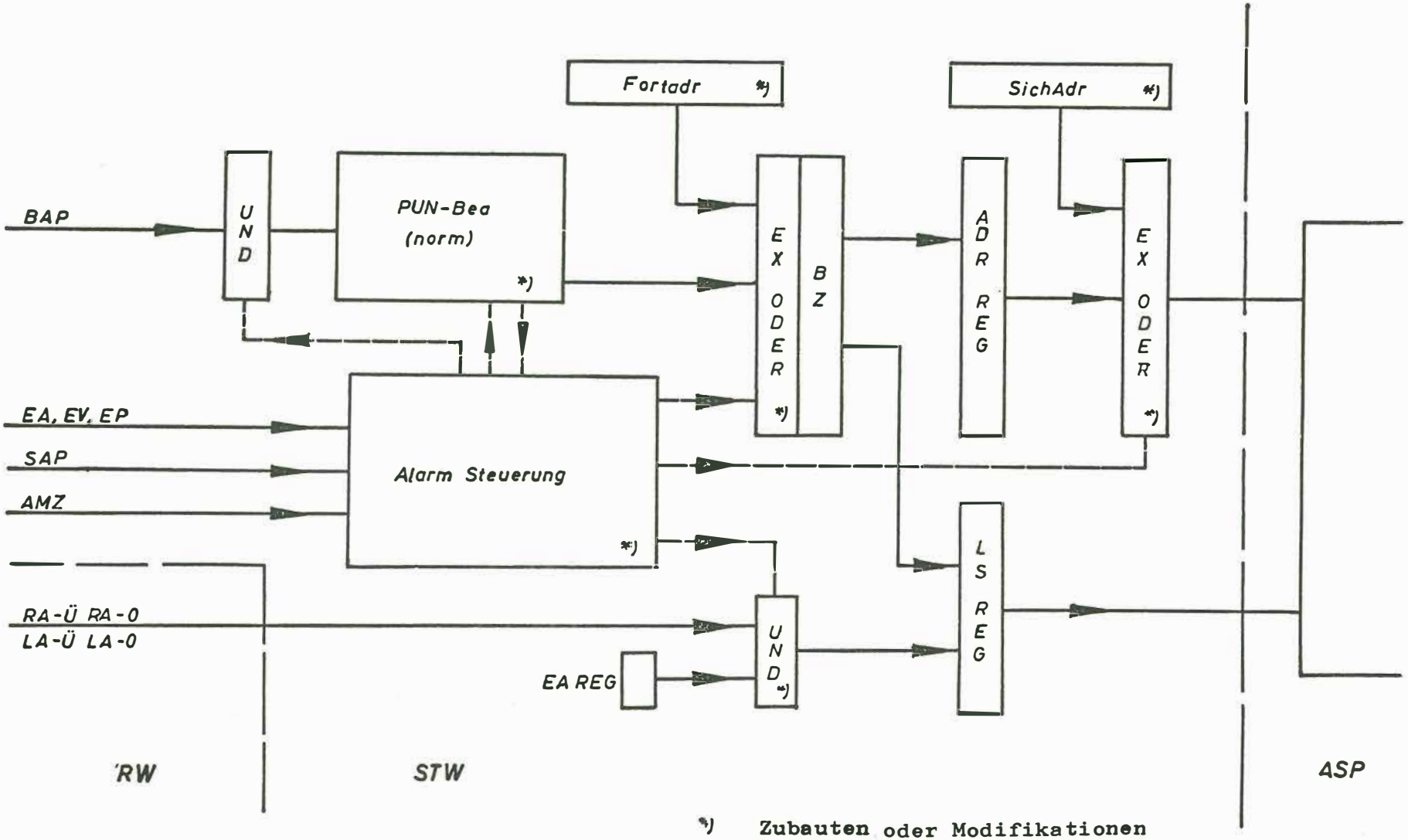


Bild 3

Vereinfachtes Blockbild des Steuerwerks mit den Erweiterungen

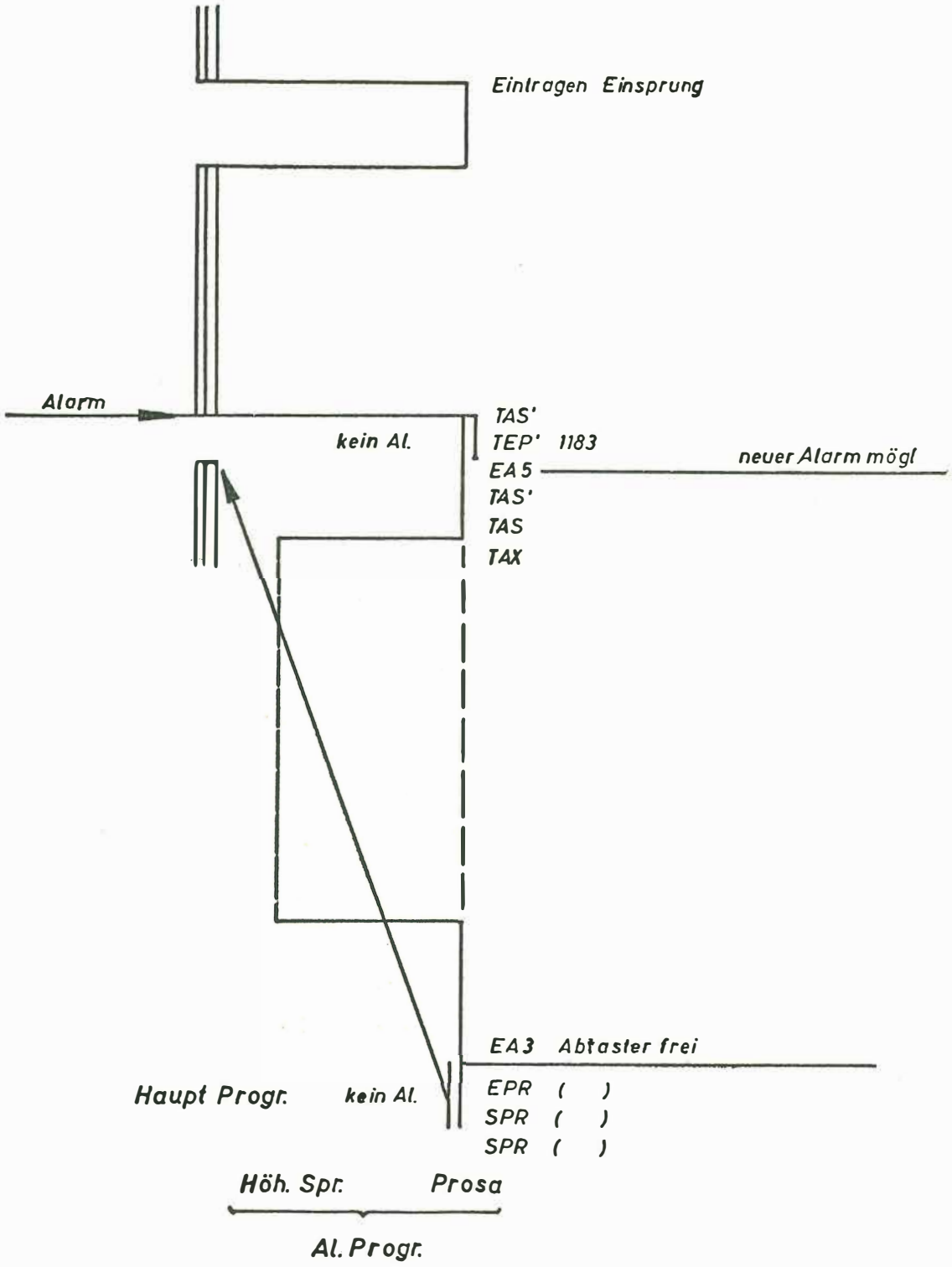


Bild 4

Alarmprogramm in einer anwendernahen Sprache

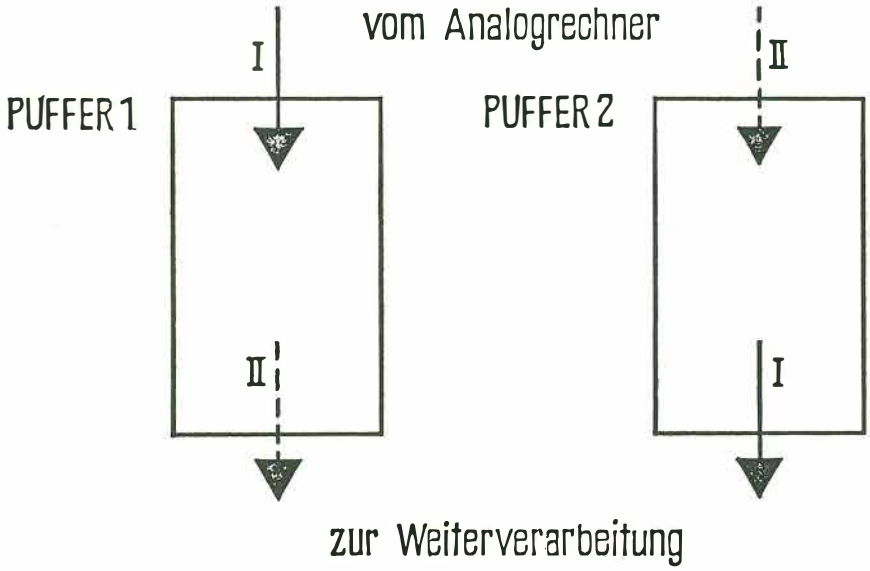


Bild 5
Schema des kontinuierlichen Datenverkehrs über einen Rechner

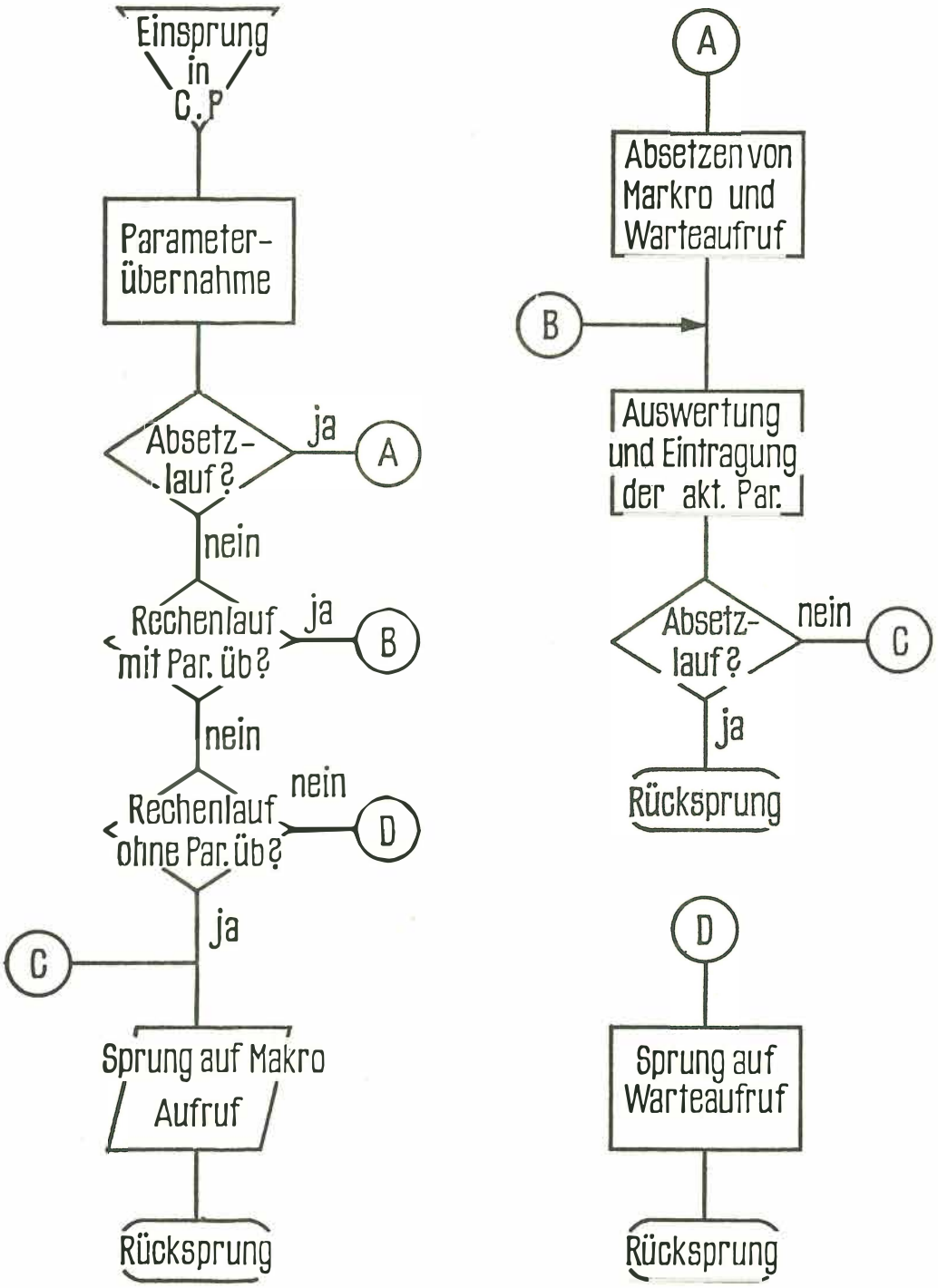


Bild 6

Schema einer Codeprozedur zur Simultanarbeit von Externverkehr und auslösendem Programm in einer anwendernahen Sprache