K.-H. Krause
Siemens AG Erlangen

Grundsatzsoftware des PR 320

Assembler 320

Die Assemblersprache des PR 320 ist vollkommen neu gegenüber der der anderen Maschinen des Systems 300.
Dies hat zwei Gründe. Einmal war man durch die neue Befehlsstruktur sowieso gezwungen eine neue Sprache zu schaffen, zum anderen wollte man eine Lücke schließen, die zwischen den üblichen symbolischen und den höheren Programmiersprachen klafft. Dies ist für das hauptsächliche Einsatzgebiet des PR 320 von Bedeutung.

Gerade für den Einsatz als Geräterechner bzw. als Ersatz für verdrahtete Steuerung wo relativ intensiv das einzelne Datum, sowohl intern als auch im Austausch mit spezieller Peripherie, manipuliert wird, das jeweilige Programm aber trotzdem nicht sehr rechenintensiv ist, wird eine Sprache benötigt, die bei geringem Codieraufwand eine umfassende und effektive Datenmanipulation durch die volle Ausnutzung der HW-Eigenschaften der Maschine ermöglicht.
Außerdem sollte die Sprache den Vorteil bieten, sehr leicht erlernbar zu sein. Aus diesen Gründen läßt sich schon die Forderung ableiten, daß man bei der großen Anzahl von Hardwarebefehlen nicht für jeden Befehl eine eigene mnemotechnische Verschlüsselung schreiben muß.
Die Matrixbefehlliste bietet sich dazu an, erst einmal zwischen der eigentlichen Operation und dem Format zu trennen. Dann müßte sich der Programmierer nur die Verschlüsselungen für die Grundbefehle und für das Formatkennzeichen merken. Statt einer Anzahl von N+M-Möglichkeiten wären dann nur noch N+M-Möglichkeiten zu merken.
Um jedoch die Lücke zu den höheren Programmiersprachen zumindest bezüglich der Notation zu schließen, wird noch einen Schritt weiter gegangen.

Der Grundbefehl wird nicht mehr durch eine mnemotechnische Verschlüsselung dargestellt, sondern durch ein Operatorsymbol. Die zum Teil noch benötigte Zusatzinformation über die Operation, die bei den höheren Programmiersprachen in den Operanden steht, (ob z.B. Gleitpunkt oder Festpunktaddition) wird durch eine Typangabe am Anfang einer Zeile hinterlegt. Befehle können mit Ausnahme gewisser Einschränkungen beliebig in einer Befehlszeile aneinander gekettet werden. Auf diese Weise erreicht man eine Notierung, die der einer höheren Programmiersprache ähnlich ist. Es gibt allerdings keine Klammerung; d.h. die Befehlszeile wird sequentiell von links nach rechts abgearbeitet.

Der Hauptvorteil des Assemblers, nämlich die Transparenz bezüglich des Absetzens der Befehls bleibt jedoch voll vorhanden. Der Programmierer kann also anhand der Befehlszeile eindeutig erkennen, welche Hardware-Befehle abgesetzt werden.

Beispiele für Assemblerzeilen sind in Bild 1 dargestellt.

In Bild 2 wird von einem Beispiel ausgehend dargestellt, aus welchen Elementen man eine Assemblerbefehlszeile aufbauen kann. Diese Aufstellung ist nicht vollständig, sondern soll nur das Prinzipielle erläutern. Natürlich sind nicht alle beliebigen Kombinationen zugelassen, da man sich an die Einschränkungen halten muß, die von der Hardware gegeben sind.

Bild 3 zeigt ein kleines Programmierbeispiel, die einfache Notierung. Es werden hier mehrere Zahlen aufeinander addiert. Das hier gezeigte Beispiel stellt kein vollständiges Programm dar, da die organisatorischen Anweisungen fehlen.

Ich glaube, daß es mir anhand der gebrachten Beispiele gelungen ist, Ihnen einen Eindruck von Vorzügen der Assemblersprache der 320 zu vermitteln.

Zusätzlich dazu gibt es noch die Makrosprache, die ebenfalls vollkommen neu ist. Der Makrogenerator arbeitet als reiner Zeichenumformer, der durch bestimmte Sonderzeichen gesteuert wird. Da mit diesem Generator jedes einzelne Zeichen manipuliert werden kann, ist ein sehr breites Anwendungsgebiet gegeben. Da die Ausgabenzeichenfolge die Eingabezeichenfolge für den Assembler ist, können beide vollkommen getrennt ablaufen, wenn kein ausreichend großer Speicherplatz zur Verfügung steht. Die gesamte Übersetzung eines Makroprogrammes würde dann 3 Durchläufe in Anspruch nehmen. Bei größeren Arbeitsspeicherausbauten kann man den Makrogenerator direkt vor den Assembler schalten und kommt so mit 2 Durchläufen aus.

Organisationsprogramm

Die Betriebsmittel die eine Rechenanlage zur Verfügung stellt, müssen von einem Organisationsprogramm verwaltet und den einzelnen Anwendeprogrammen nach bestimmten Kriterien zugeteilt werden.

Diese Betriebsmittel sind:

a) Arbeitsspeicherplatz
b) Zentralprozessorzeit
c) Geräte.

Eine automatische Speicherplatzeinteilung bei reinen Arbeitsspeichermaschinen ist im Prozeßrechnerbetrieb nicht von Bedeutung, da sie ständig mit dem gleichen Programmsystem arbeiten. Eine Arbeitsspeicherverwaltung ist deshalb nur beim Laden des Systems von Bedeutung. Deshalb ist eine solche Verwaltung nicht standardmäßig vorgesehen, sondern kann als Option in das Organisationsprogramms eingebaut werden.

Die Hauptaufgabe des Organisationsprogrammes erstreckt sich daher auf die beiden Betriebsmittel Zentralprozessor und Geräte. Dementsprechend wird diese kurze Übersicht als Hauptpunkte die Programmorganisation und - Koordinierung und den Ein-/Ausgabeverkehr umfassen. Abschließend wird noch kurz auf die Struktur und die Generierung eingegangen.

1. Programmorganisation

Als Prozeßrechner hat die PR 320 selbstverständlich ein Echtzeitorganisationsprogramm vo mehrere Programme den Zentralprozessor entsprechend ihrer Priorität benutzen können. Das jeweilige höchstpriore ablauffähige Programm bekommt ihn zugeteilt.

Die Anzahl der Programme ist vom logischen her lediglich begrenzt durch die Eindeutigkeit der Identifikation d.h. der Programmnummer und beträgt 65535. Praktisch ist die Anzahl der Programme durch den zur Verfügung stehenden Arbeitsspeicherplatz begrenzt. Da ja jedes Programm einen eigenen Programmkopf, die sogenannte Parametertafel mit 64 Worten Länge besitzt, ergibt sich daraus bei einem Adressiervolumen von 64 KW eine Grenze bei 1000 Programmen.

Die Programmpriorität ist von der Programmidentifikation, der Programmnummer also, vollkommen getrennt. Insgesamt gibt es 255 Prioritäten. Daraus folgt, daß mehrere Programme die gleiche Priorität besitzen können. Aufgrund dieser Trennung von Priorität und Identifikation ist es leicht möglich, die Priorität dynamisch zu wechseln. Dadurch ist es möglich, sich optimal dem jeweiligen Prozeßzustand anzupassen. Außerdem ist es noch möglich ein Programm in mehrere Aufgaben - maximal 48 - zu unterteilen. Diese Aufgaben besitzen innerhalb eines Programmes unterschiedliche Prioritäten. Die Aufgabennummer ist identisch mit der Priorität.

Die Aufgaben eines Programmes können nicht simultan zueinander laufen. Sie werden entsprechend ihrer Priorität nacheinander abgearbeitet. Sind alle Aufgaben eines Programmes abgearbeitet, so ist auch das Programm beendet.

Neben Programmen gibt es noch Codestücke, die keinen eigenen Befehlszähler besitzen, sie werden jeweils mit dem Befehlszähler des aufrufenden Programmes durchlaufen.

Dieses Codestück kann dabei von beliebig vielen Programmen aufgerufen und simultan durchlaufen werden; dies bedeutet, daß dieses Programmstück reentrant geschrieben sein muß. Dies wird durch die Tatsachen ermöglicht, daß ASP-Zellen über Register adressiert werden können und jedes Programm seinen eigenen Registersatz besitzt. (Siehe Bild 4). Dieses Programmstück kann einmal ein Unterprogramm sein. Dann müssen aber alle aufrufenden Programme dessen Adresse kennen. Deshalb ist im Organisationsprogramm eine Funktion realisiert mit deren Hilfe man solche Programmstücke per Namen aufrufen kann. Diese werden dann Common-Codes genannt.

2. Programmkoordinierung

An Koordinierungsaufrufen stehen zur Verfügung:

- Starten
- Beenden
- Anhalten
- Fortsetzen
- Warten

Zusätzlich hat man noch eine Funktion geschaffen, mit deren Hilfe es dem Anwender möglich ist, beliebige freie Warteschlangen zu bilden. Diese Funktion wird Koordinierungszähler genannt. Er wird durch einen Aufruf eingerichtet und hat folgende Bestimmungsgrößen:

- Name
- obere Grenze oG
- untere Grenze uG
- Zählvariable V

sowie eine Modusangabe, die die Bearbeitung einer ihm zugeordneten Warteschlange regelt. Die Zählvariable V kann man nun per Aufruf in Schritten von 1 erhöhen oder erniedrigen. In Abhängigkeit davon, welchen Wert die Zählvariable nach der Operation annimmt, werden verschiedene Funktionen ausgeführt (Bild 5).
Es sind damit beliebige Programmkoordinierungen möglich.
Die meisten Aufrufe können derart modifiziert werden, daß sie zeitverzögert erfolgen. Der Start kann außerdem noch zyklisch ausgeführt werden.

3. E/A-Verkehr

3.1 Generelle Aufrufstruktur

Jeder Aufruf - gleichgültig ob ein interner Koordinierungsaufruf oder ein E/A-Aufruf - beginnt mit einem Ruf an das Organisationsprogramm. Da das Organisationsprogramm in einem eigenen HW-Zustand d.h. in 2,Ø läuft, ist dies ein eigener Befehl nämlich der Befehl "Rufen Primärzustand". Dabei muß ein vereinbartes Register mit der Adresse eines Parameterfeldes geladen werden. In diesem Parameterfeld - Parameterblock (PB) genannt - ist die auszuführende Operation genau spezifiziert. Ist die auszuführende Funktion eine E/A-Operation, so ist in dem Parameterblock noch der Verweis auf ein weiteres Parameterfeld - genannt Geräte - Datei-Feld (GEDA-Feld) - eingetragen. In diesem GEDA-Feld ist das Gerät genauer beschrieben. Beziehen sich mehrere Aufrufe eines Programmes auf ein Gerät, so genügt es in der Regel einmal ein GEDA-Feld anzulegen. Den prinzipiellen Aufbau dieser Parameterfelder zeigt Bild 6.

3.2 Koordinierung des E/A-Verkehrs

Oft ist unvorteilhaft sich schon bei der Programmierung auf ein bestimmtes Gerät festlegen zu müssen. Deshalb hat der Programmierer die Möglichkeit symbolische Gerätenamen anzugeben, denen dann beim Programmanlauf ein bestimmtes Gerät zugeordnet wird.
Die eigentliche Koordinierung, d.h. der Zugriff zu den Geräten selbst, erfolgt in einem zweistufigen Verfahren.

Der Koordinierung der Aufruffolge über die Transferwarteschlange, die dafür sorgt, daß eine E/A-Operation nach der anderen abläuft, ist ein Belegmechanismus übergeordnet. Erst wenn ein Programm aufgrund seiner Stellung in der Belegwarteschlange mit dem Gerät verkehren darf, können von ihm Transferaufrufe abgegeben werden. Um möglichst flexibel zu sein, hat man drei verschiedene Belegmodi eingeführt,nämlich:

- geteilt Belegen
- exklusiv Belegen
- mit Vorrang Belegen.

Die beiden ersten Modi bedürfen keiner Erklärung. Die Vorrangbelegung bedeutet, daß das entsprechende Programm auf jeden Fall mit dem Gerät verkehren darf. Der Belegmechanismus wird in diesem Fall übersprungen und die Aufrufe des Programmes können gleich in die Transferwarteschlangen eingetragen werden.

Die Einträge in die Transferwarteschlangen selbst, können entweder nach Priorität oder nach der der zeitlichen Reihenfolge des Eintreffens vorgenommen werden. Um den Programmierer von unnötiger Arbeit zu entlasten, kann das Belegen auch implizit geschehen. Die Belegung erfolgt dann in einem gerätespezifischen Standardmodus.

4. Struktur und Generierung

Das gesamte Organisationsprogramm ist funktionsmodular aufgebaut, d.h. nur eine gewisse Grundmenge der Funktionen ist zwingend vorgeschrieben, alle übrigen können aufgrund von Wünschen eingebaut werden.

Das Organisationsprogramm ist also bezüglich seiner Geräteausstattung und bezüglich seiner Funktionen generierbar. Das Generieren geschieht beim Laden.
Dies bedeutet, daß der Kunde immer den kompletten Masterstapel zur Verfügung gestellt bekommt. Der Kunde legt dann vor diesen Masterstapel seine Wünsche in Form von Parameterkarten. Beim Einlesen des Stapels bzw. der Lochstreifenrollen wird dann das gewünschte Organisationsprogramm zusammengestellt. Dies hat den großen Vorteil, daß bei Änderung der Funktionen bzw. der E/A-Ausstattung nicht immer ein neues Organisationsprogramm bestellt werden muß.

Beispiele für Assemblerzeilen

Befehlszeilen:

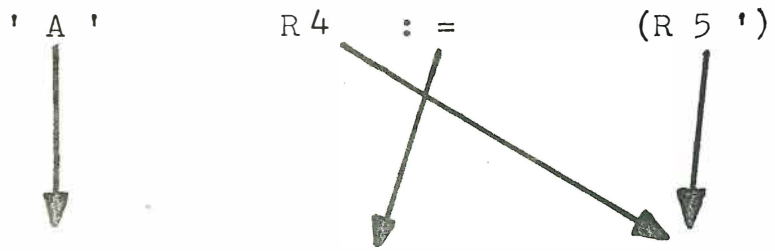
```
' IA '      R 4 : = (ALPHA + R 7) - 735/R 5 - (BETA)
' F  '      R 5 > = (GRENZWERT) : SP FEHLER
' B  '      (OTTO (7)) : T 1
```

Datenzeilen:

```
' VF '      WERT/ = 15, - 87,, 4 <ø >, 'H = FF'
            TEXT/ = ' Z = ZEIT :  . .  1972 '
```

Bild 1

Möglicher Aufbau einer Befehlszeile:



A (Betrag)	+.	Konstanten
	-	und Adressen (C, AI, RX)
F (Festpunkt)	x	
G (Gleitpunkt)	/	Ri R
B (Bit)	: =	(Ri) A
.	: T	(Ri') A I
.	. E	('Ri) D A
.	.	(ADR) AX
		(ADR + Ri)

Bild 2

Programmierbeispiel

```
' IA '      R 3 : = 27
            R 4 : = Ø
            R 5 : = WERTADR


'F' Schleife/ R 4 : = R 4 + (R 5')
              R 3 : DS SCHLEIFE

    (Summe)       : = R4
          .
          .
          .
          .

'VF' WERTADR/ = 1,2,3,1Ø < - 35 > 4,5,6,7,18
              = 9 <432>

     SUMME/   =
              .
              .
```

Bild 3

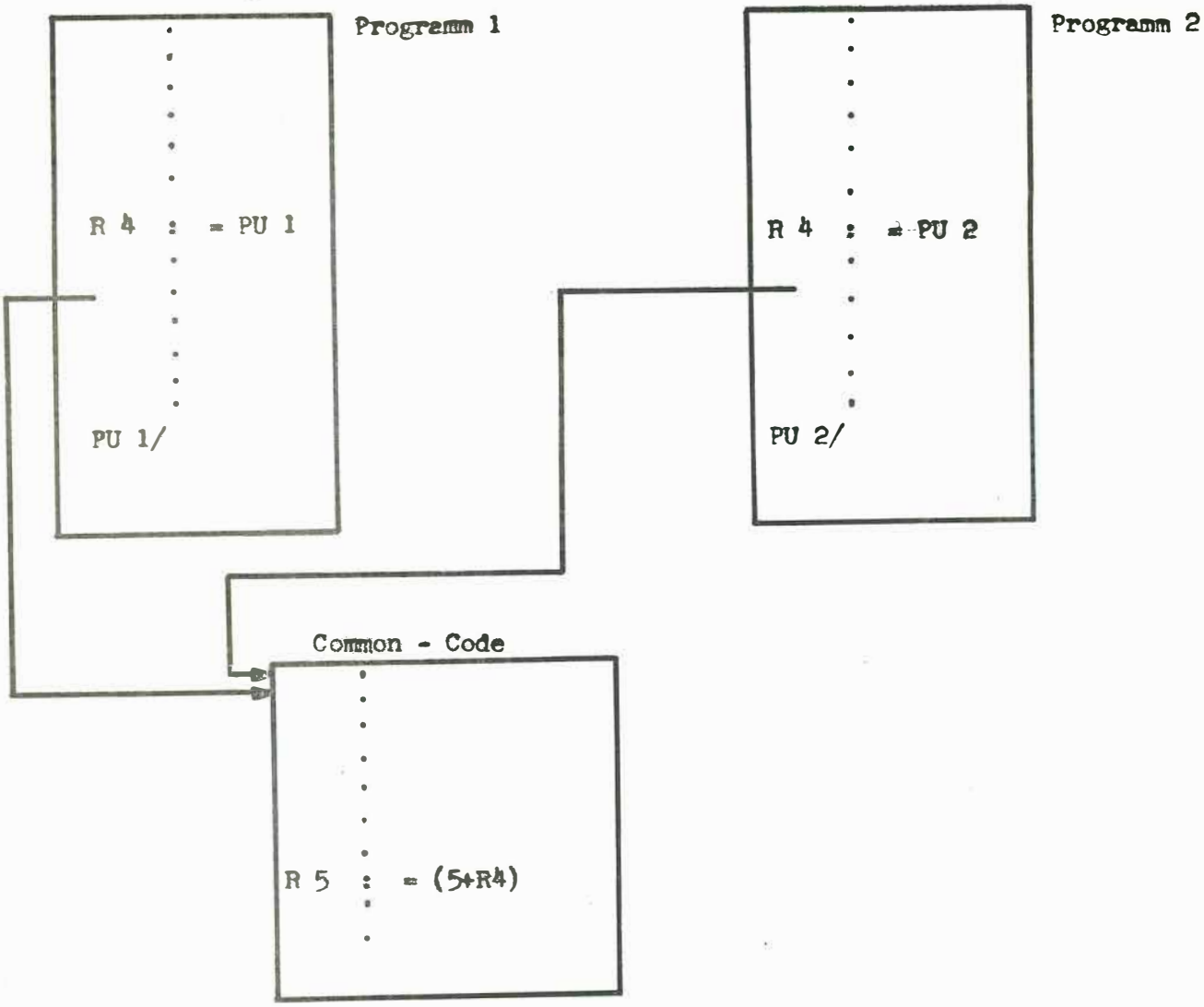


Bild 4

Wirkungsweise des Koordinierungszählers

Zähler erhöhen:

$V \leq uG$	$uG < V \leq oG$	$V > oG$
Warteschlange fortschalten	keine Wirkung	Programm absetzen und in die Warteschlange eintragen

Zähler erniedrigen:

$V < uG$	$uG \leq V < oG$	$V \geq oG$
Programm absetzen und in WS eintragen	keine Wirkung	Warteschlange fortschalten

Bild 5

Aufrufstruktur

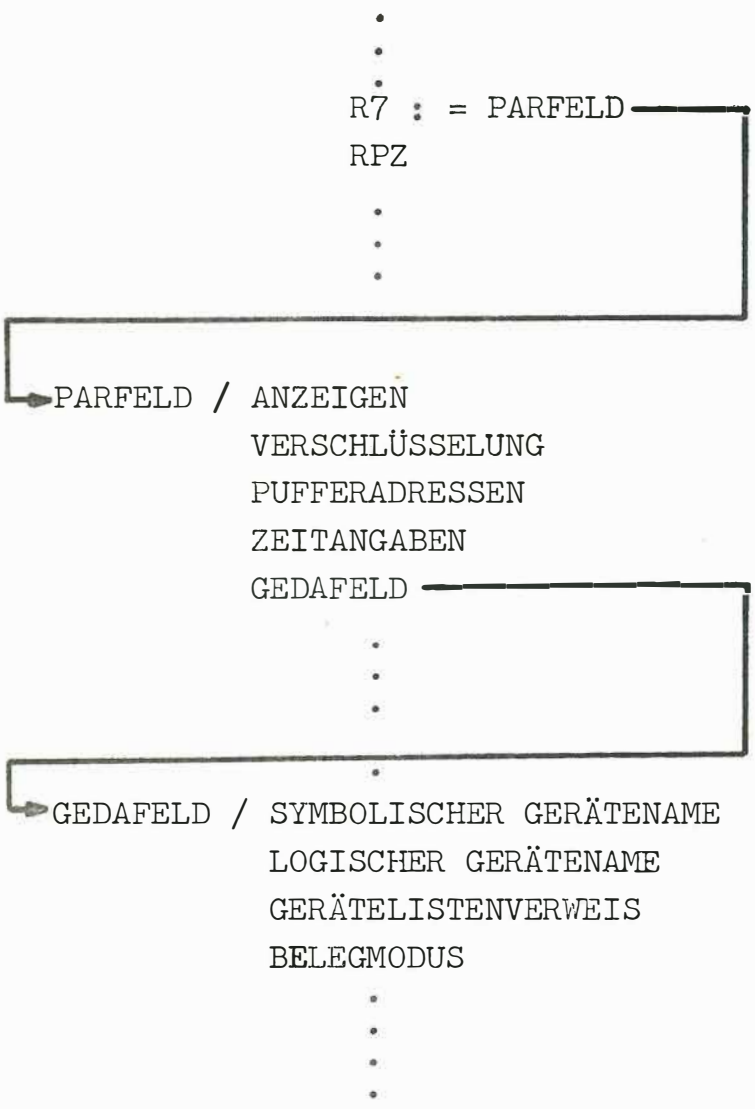


Bild 6