PDV-E 142
Juli 1980

PDV-Entwicklungsnotizen

Codegenerator- und Systembeschreibung der SIEMENS 310 — PEARL-Implementation

M. Trautner
Physikalisches Institut der
Universität Erlangen-Nürnberg

Kernforschungszentrum Karlsruhe

PDV-Berichte

Die Kernforschungszentrum Karlsruhe GmbH koordiniert und betreut im Auftrag des Bundesministers für Forschung und Technologie das im Rahmen der Datenverarbeitungsprogramme der Bundesregierung geförderte Projekt Prozeßlenkung mit Datenverarbeitungsanlagen (PDV). Hierbei arbeitet sie eng mit Unternehmen der gewerblichen Wirtschaft und Einrichtungen der öffentlichen Hand zusammen. Als Projektträger gibt sie die Schriftenreihe PDV-Berichte heraus. Darin werden Entwicklungsunterlagen zur Verfügung gestellt, die einer raschen und breiteren Anwendung der Datenverarbeitung in der Prozeßlenkung dienen sollen.

Der vorliegende Bericht dokumentiert Kenntnisse und Ergebnisse, die im Projekt PDV gewonnen wurden.

Druck und Verbreitung:

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640 7500 Karlsruhe 1

Bundesrepublik Deutschland

PDV

PDV-E 142

PROJEKT PROZESSLENKUNG MIT DV-ANLAGEN
ENTWICKLUNGSNOTIZ PDV-E 142

CODEGENERATOR- UND SYSTEMBESCHREIBUNG DER 310 - PEARL-IMPLEMENTATION

VON
M. TRAUTNER
PHYSIKALISCHES INSTITUT DER
UNIVERSITAET ERLANGEN-NUERNBERG

47 SEITEN

JULI 1980

Codegenerator- und Systembeschreibung

der 310 - PEARL-Implementation

M. Trautner

Physikalisches Institut der Universität Erlangen-Nürnberg

Juli 1980

Inhaltsverzeichnis

	Seite
1. Übersicht über das PEARL-Cross-Compiliersystem	1
1.1. Übersetzungssystem	1
1.2. Kurzbeschreibung der neu erstellten Systemkomponenten	1
1.3. Sprachumfang	3
2. Code-Abbildung und Schnittstellenbeschreibung	4
2.1. SchnittstellenPEARL-Modul/PEARL-Betriebssystem	4
2.2. Programmstruktur	1Ø
2.3. Referenzierung	22
2.4. Parameterübergabe	25
2.5. Tasking und Synchronisation	27
2.6. Signalbearbeitung	29
2.7. Algorithmische Funktion	30
2.8. Prozeß-E/A	35
2.9 Standard-E/A	40
3. Laufzeitprozeduren	44
3.1. Allgemeines	44
3.2. Kurzbeschreibung der Standardfunktionen	45
3.3. Besonderheiten der Systemprozeduren	46
4. Literaturverzeichnis	47

1. Übersicht über das PEARL-Cross-Compiliersystem für die S31Ø

Das Cross-Compiliersystem wurde von der Gruppe Datenverarbeitung des Physikalischen Instituts der Universität Erlangen, gefördert durch das BMFT/PDV, entwickelt /1/. Ein wesentlicher Aspekt dabei war die Erprobung portabler PEARL-System-Software /2/. Der Aufwand für die Erstellung des gesamten Systems konnte hierdurch gegenüber den vergleichbaren Komponenten einer früheren Implementation für die S3Ø6 /3/ von 6 auf 2,5 Mannjahre gesenkt werden.

1.1. Das Übersetzungssystem

PEARL-Quellmodule werden auf der 3Ø6 vom sogn. Oberen Compilerteil /4/ in die Zwischensprache CIMIC/1 /5/, diese von einem Codegenerator in 3ØØ-Assembler und dann vom Cross-Assembler ASSB in Grundsprache übersetzt. Über Teletype-Kopplung werden die Grundsprachedateien von der 3Ø6 zur 31Ø auf Floppy-Disk überspielt. Ein Hauptmodul und gegebenenfalls Prozedurmoduln werden dort durch den Ladebinder BL1Ø mit den benötigten Elementen einer PEARL-Laufzeitbibliothek zu einem Programm gebunden, dessen Start und Ablauf durch ein PEARL-Betriebssystem gesteuert wird.

1.2. Kurzbeschreibung der neu erstellten Systemkomponenten

1.2.1. Codegenerator

Der Codegenerator besteht aus vier Läufen, die zusammen 25% der Übersetzungszeit benötigen (Oberer Compilerteil: 50%, Assembler: 25%).

Aufgabe der einzelnen Läufe:

a) Umsetzung von CIMIC/1 in einen Zifferncode (Programm CIZI /6/)
b) Eigentliche Code-Erzeugung (mehrere Dateien) (Programm CG1Ø)
c) Binden von Dateien mit Zeichen-Codeumsetzung (von Siemens-Interncode nach ASCII), (Programm CGBI)
d) Erzeugung eines Stackmoduls, dabei werden für jede Task der Gesamtbedarf an lokalen Daten errechnet, der bestimmt wird durch den Eigenbedarf und den Bedarf der von ihr aufgerufenen Prozeduren (Programm CGBI).

1.2.2. Datenübertragung 3Ø6 - 31Ø

Die Grundsprachedateien werden über eine Teletypeleitung mit 48ØØ baud übertragen; hardware-Schnittstelle bei der 3Ø6 ist ein CAMAC-TTY-Einschub, bei der 31Ø eine Sichtgeräteanschaltung im TTY-Betrieb. Die Kopplungsprogramme sind jeweils in PEARL geschrieben und führen die Übertragung in Blöcken nach einem Protokoll gemäß DIN 66Ø19 durch, wobei fehlerhafte Übertragungen erkannt und nach negativer Quittung wiederholt werden können.

1.2.3. Das PEARL-Betriebssystem (PBS)

Das PBS /7/ /8/ verwaltet die Ein/Ausgabe bei den "langsamen" E/A-Geräten, bearbeitet die Tasking- und Semaphoroperationen (durch Ein- und Ausketten von Tasks in Warteschlangen und Retten der Registertafel) und übernimmt die Erkennung und Bearbeitung von 16 Interrupts, die über eine alarmbildende Digitaleingabe erzeugt werden. Es erlaubt beliebig viele Tasks, die beim Aktivierungsaufruf mit einer aktuellen Priorität versehen werden können.

Das PBS ist ein Anwenderprogramm (unter ORG-Verwaltung) und belegt die Programmebenen 13 und 14. Aufträge von PEARL-Tasks (alle Ebene 15) und Rückmeldungen nach einem ORG-Aufruf laufen über eine einzige Schnittstelle (Koordinierungszählererhöhung). Das PBS besteht aus einem rechnerunabhängigen Verwaltungsteil (in einem virtuellen höheren Assembler geschrieben) und aus einem anlagenabhängigen Teil (in AS3ØØ formuliert), Gesamtlänge: 18ØØ Speicherworte.

1.2.4. Laufzeitsystem

Die Prozeduren des LZ-Systems sind reentrant formuliert, sodaß jede Prozedur - wenn überhaupt - nur einmal im Hauptspeicher stehen muß, auch wenn sie von mehreren Tasks gleichzeitig aufgerufen wird. Die so gewonnene Reduzierung des Speicherplatzbedarfs überwiegt bei weitem die gegenüber nichtreentrant formulierten Prozeduren gering erhöhte Laufzeit.

Den platzmäßig größten Teil des LZ-Pakets stellen die Formatierungsroutinen für die Standard-E/A /9/ dar (~ 7K Worte), die in PEARL formuliert und somit anlagenunabhängig sind. Alle anderen LZ-Prozeduren (Prozeß-E/A, Zeichenkettenverarbeitung, Schnittstellenanpassroutinen, arithmetische Routinen u.ä.) sind in Assembler geschrieben und belegen zusammen nur etwa 1,4 K Worte.

1.3. Sprachumfang

Der für die 31Ø verwirklichte Sprachumfang ist, bis auf das Fehlen der graphischen EA vergleichbar mit dem früher für die 3Ø6 verwirklichten PEARL-Subset /10/ (gleicher Compileroberteil). Letzterer wiederum entspricht funktionell (also von syntaktischen Unterschieden abgesehen) dem DIN-Normvorschlag für BASIC-PEARL /11/, lediglich Strukturen sind noch nicht eingeführt.

Besonderheiten der Implementation:

Anzahl der Tasks beliebig
grundsätzlich reentrantfähige Anwenderprozeduren
getrennte Übersetzbarkeit von Hauptmodul (mit Tasks und Prozeduren) und Prozedurmoduln (nur Prozeduren).

2. Code-Abbildung und Schnittstellenbeschreibung

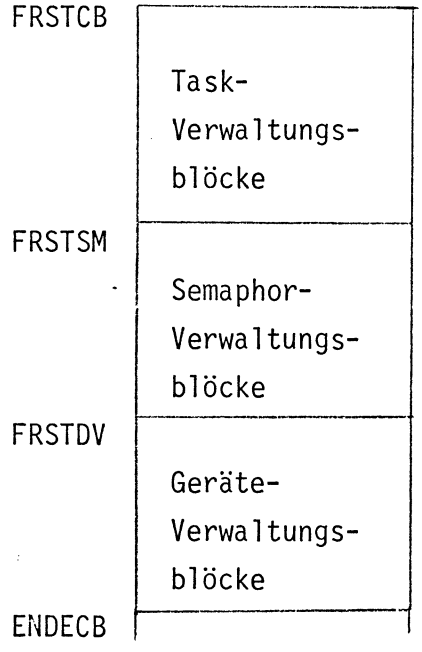
2.1. Beschreibung der Schnittstelle PEARL-Modul zum PBS-31Ø

Die Schnittstelle PEARL-Modul zum PEARL-Betriebssystem der S31Ø besteht aus:

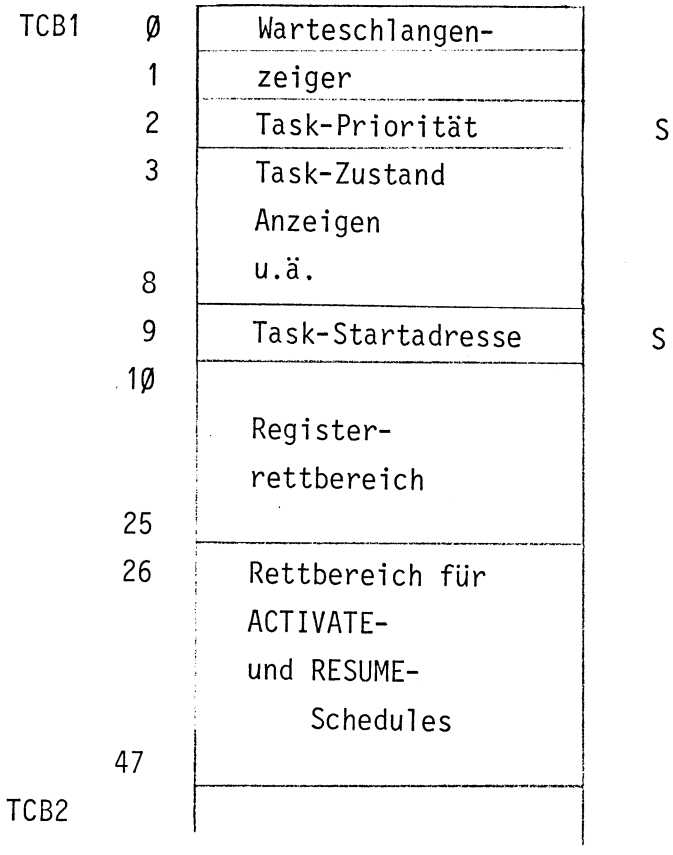
- Task-, Semaphor- und Geräte-Verwaltungsblöcken
- Koordinierungszähler, "Briefkasten", Auftragsparameterblöcken
- Initialisierung bei Systemstart

2.1.1. Verwaltungsblöcke

Die Verwaltungsblöcke, im folgenden oft auch als "Kontrollblöcke" bezeichnet, sind vom Code-Generator anzulegen. Zellen, die mit Information vorzubelegen sind, werden in den folgenden Darstellungen mit S (statistisch) bzw. D (dynamisch) gekennzeichnet. Die Verwaltungsblöcke sind unmittelbar hintereinander, gruppiert und in der folgenden Reihenfolge anzulegen: (Die Gruppenanfangsadressen und die Adresse der 1. Zelle nach den Verwaltungsblöcken werden bei der Initialisierung dem PBS übergeben, sie sind in der Darstellung durch symbolische Adressen gekennzeichnet).



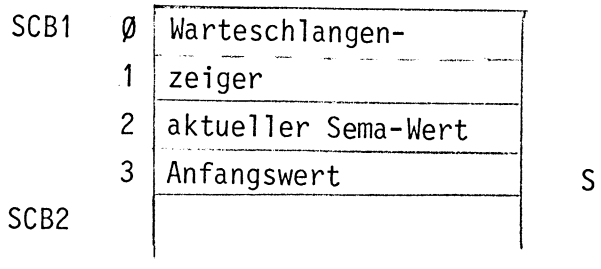
2.1.1.1. Aufbau eines Taskverwaltungsblocks, 48 Worte



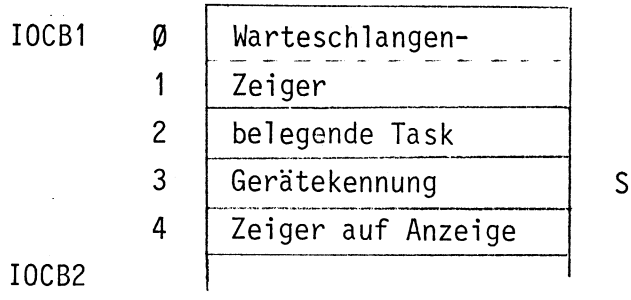
Bemerkung:

Diejenige Task, deren Verwaltungsblock an erster Stelle steht, wird vom PBS31Ø als Starttask betrachtet und nach der Initialisierung sofort gestartet, wobei ihre Priorität auf Ø gesetzt wird.

2.1.1.2. Aufbau eines Semaphor-Verwaltungsblocks, 4 Worte



2.1.1.3. Aufbau eines Geräte-Verwaltungsblocks, 5 Worte



Bisher sind folgende Gerätekennungen vereinbart:

Ø = RENA-Drucker (BS2)
1 = BS-Ausgabe (BSØ)
2 = BS-Eingabe (BSØ)
3 = Analog-Eingabe

2.1.2. Aufträge des PEARL-Moduls an das PBS31Ø

Die Aufträge erfolgen durch Ablegen der Adresse eines Auftragsparameterblocks in einem "Briefkasten" und anschließende Erhöhung des Auftrags-Koordinierungszählers (ORG-Aufruf); dadurch kann das wartende PBS31Ø weiterlaufen, während das niederpriore PEARL-Programm in einen Wartezustand versetzt wird.

2.1.2.1. Vereinbarung von Briefkasten und Koordinierungszähler

Da ein PEARL-Modul und das PBS31Ø weder gemeinsam übersetzt noch gebunden werden, werden die beiden "Kontaktstellen" durch absolute Adressen an geeigneter Stelle (z.B. direkt unterhalb des ORG) festgelegt.
Die Adressierung auf Assemblerebene erfolgt günstigerweise symbolisch nach vorheriger 'IDENT'-Anweisung (wegen leichter Wartbarkeit bei Änderung).
Länge des Briefkastens: 1 Wort
Länge des Koordinierungszählers: 3 Worte
Der Koord.-Zähler wird bei der Initialisierung vom PBS31Ø eingerichtet und geeignet initialisiert.

2.1.2.2. Aufbau der Auftragsparameterblöcke

Parameterblöcke unterteilen sich anhand ihrer Länge in drei Gruppen:

- allgemeiner Parameterblock (ohne Schedule) 4 Worte
- Parameterblock mit Schedule 11 Worte
- I/O-Parameterblock 5 Worte

Allgemeiner Parameterblock:

PARBLK			
	Ø	W.S.-Zeiger	
	1	Zeiger auf TCB oder SCB	D/S
	2	Funktionskennung	S
	3	Ø (≙ Leer-Schedule)	S

Parameterblock mit Schedule:

SCHBLK			
	Ø	W.S.-Zeiger	
	1	Zeiger auf TCB	D/S
	2	Funktionskennung	S
	3	Schedule-Typ	D
	4	'Schedule'-Priorität	[D]
	5	aktueller Typ	
	6	Z. auf Z. auf Schedule	
	7	Restzeit	
	8	Interrupt-Nummer	D
	9	'AFTER'-Intervall	D
	10	'ALL'-Intervall	D

I/O-Parameterblock:

Ø	W.S.-Zeiger	
1	Zeiger auf Geräteblock	D
2	Funktionskennung: 9	S
3	Leer-Schedule : Ø	S
4	Pufferanfangsadresse	D

Erklärungen:

- In das 2. Wort des Parameterblocks ist die Adresse desjenigen Verwaltungsblocks einzutragen, der das "Objekt" der gewünschten Funktion ist, z.B. bei ACTIVATE TASK3 der Taskverw.-Block der TASK3, nicht etwa der der aufrufenden Task ("Subjekt"). Bezieht sich ein Aufruf auf die eigene Task, so kann an dieser Stelle auch Ø stehen.

- Funktionskennungen:

  1 = ACTIVATE
  2 = REQUEST
  3 = RELEASE
  4 = TERMINATE
  5 = PREVENT
  6 = SUSPEND
  7 = CONTINUE
  8 = RESUME
  9 = I/O-FUNKTION

- Scheduletyp:

  Ø = Leerschedule
  5 = ALL
  8 = AFTER
  9 = AFTER ALL
  12 = WHEN
  13 = WHEN ALL

- Die I/O erfolgt über Puffer, deren Anfangsadressen angegeben werden müssen (von den Laufzeitprozeduren). Das PBS31Ø erwartet die Endeadresse des Puffers im 1. Wort des Puffers, erst danach folgen die Daten. Die Puffer werden als lokale Daten von I/O-Laufzeitroutinen angelegt.
  Die Platzreservierung für Auftragsparameterblöcke erfolgt im Taskkopf (siehe dazu 2.2.1.).

2.1.3. Systemstart, Initialisierung, Programmstart

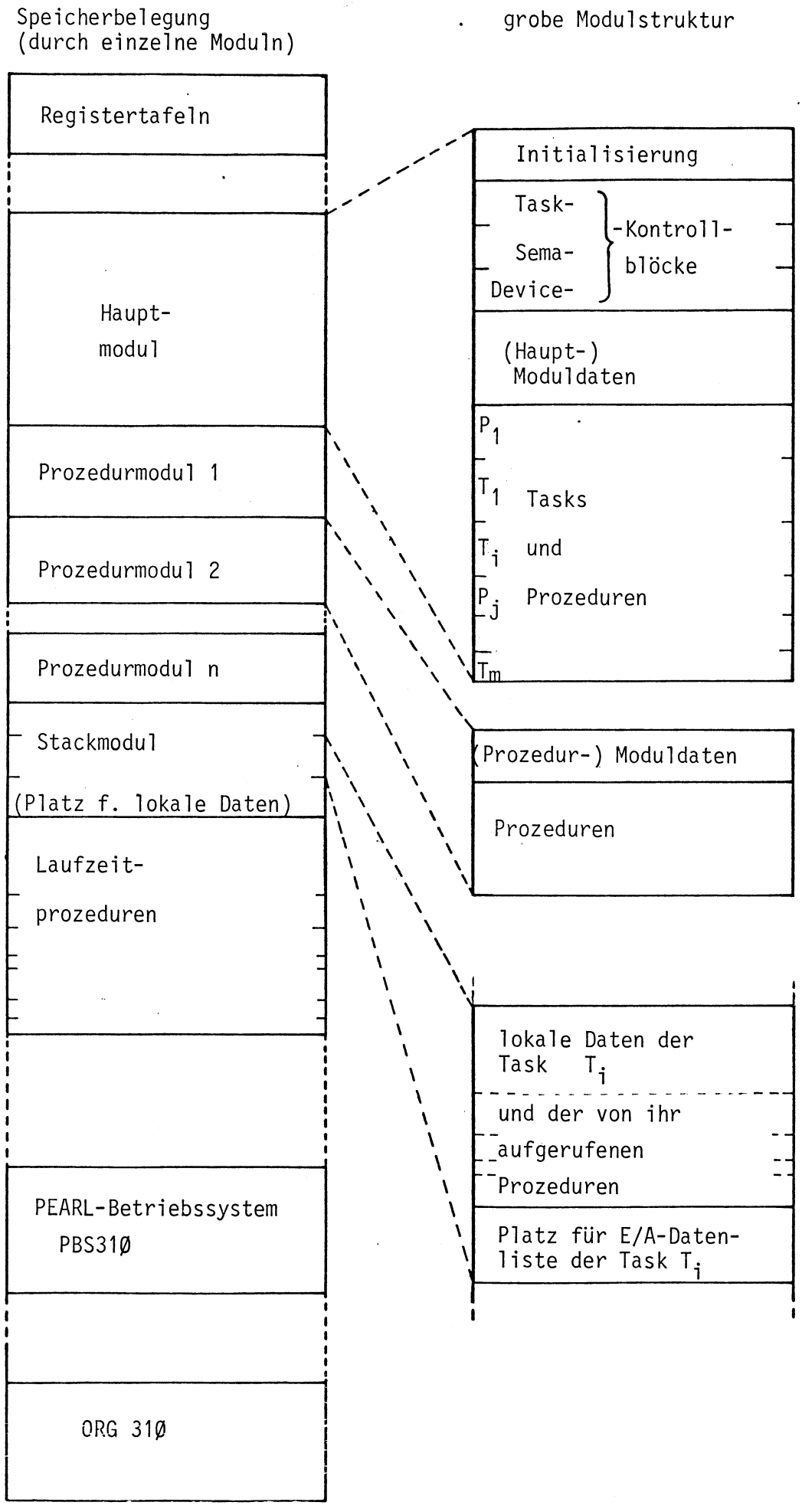
Der PEARL-Modul ist unter der Programmnummer 15, das PBS31Ø auf Ebene 14 zu laden (Ladeadressen beliebig). Der Start des Programms erfolgt durch Start des PBS31Ø. Dieses erfragt zunächst über Bedienungsanweisung, ob das Zeitgeberprogramm (im PBS) benötigt wird und startet es gegebenenfalls (auf Ebene 13). Dann richtet es Briefkasten und Koordinierungszähler mit Initialwert Ø ein, gibt an das ORG einen Startauftrag für das PEARL-Programm und erniedrigt den Koordinierungszähler. Hierdurch wird das PBS31Ø angehalten und das PEARL-Programm kann einen Initialisierungsauftrag abgegeben:
Die Adresse eines Parameterblocks wird im Briefkasten übergeben und der Koordinierungszähler per ORG-Aufruf erhöht. Im Parameterblock stehen folgende Adressen:

```
ADRBLK/ = FRSTCB  Adresse des 1. Task-Kontr.Blockes
        = FRSTSM      "    "  1. Sema-   "     "
        = FRSTDV      "    "  1. Device- "     "
        = ENDECB  Ende-Adresse der Kontrollblöcke + 1
```

Nach der Initialisierung (u.a. Vorbelegung von Kontrollblöcken) startet das PBS diejenige Task, deren Kontrollblock an 1. Stelle steht.

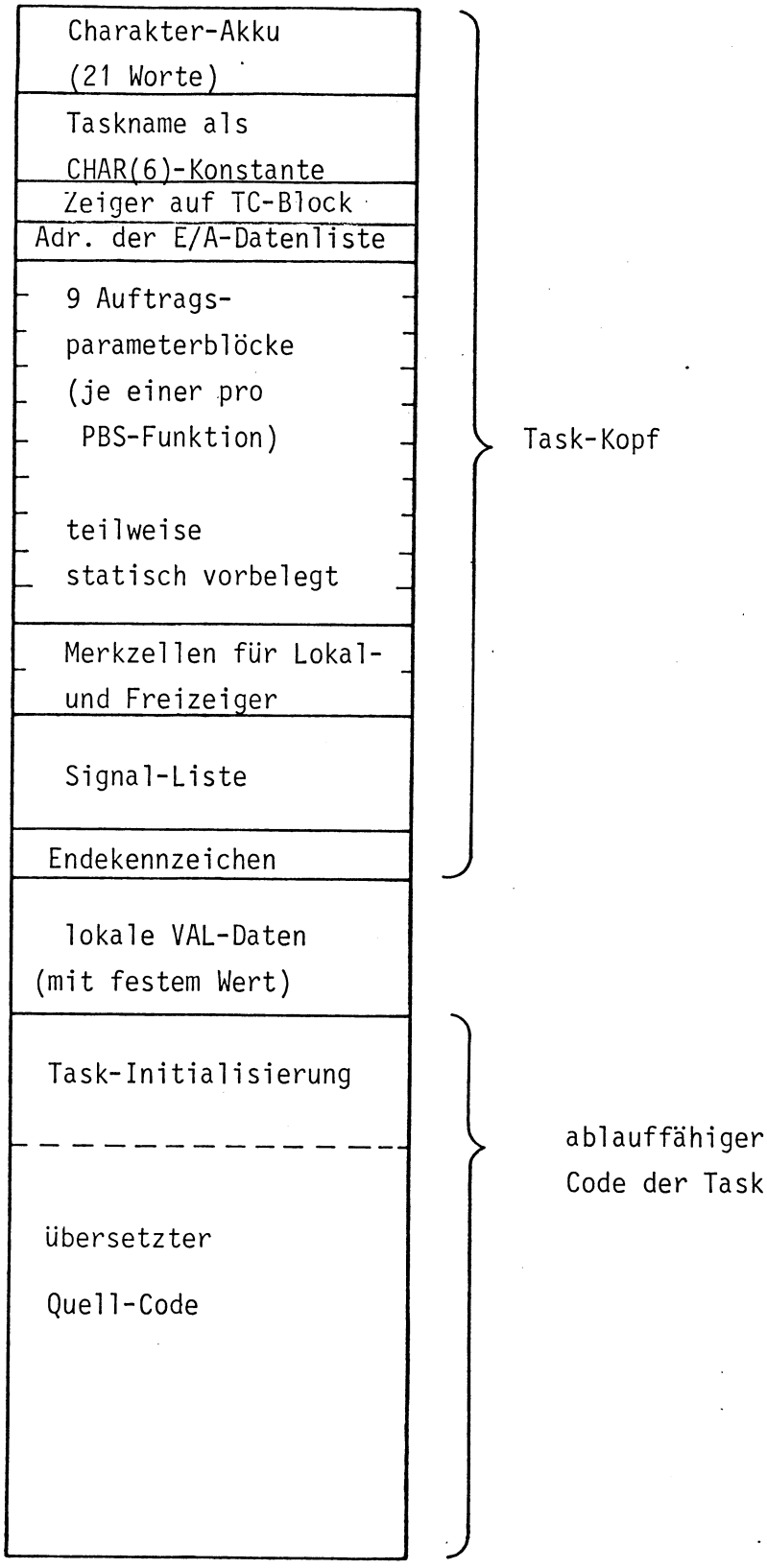
2.2. Programmstruktur

2.2.1. Speicheraufteilung



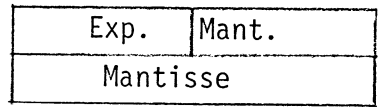
Struktur einer Task

Der Taskkopf ist eine Datenstruktur, die zur Unterstützung des Laufzeitsystems dient (ähnlich wie der Taskkontrollblock für das PBS).



2.2.2. Datenstruktur:

2.2.2.1. Einfachgrößen

FIXED	1 Wort	2'er Komplement-Darstellung
FLOAT	2 Worte	 Exp. und Mantisse im 2'er Kompl.
BIT(n)	1 Wort	n ≤ 16 Darstellung linksbündig Die Länge n wird dynamisch verwaltet
CHAR(n)	n/2 Worte	n ≤ 4Ø linksbündig wenn n ungerade ⇒ letztes Byte = Ø Die Länge wird dynamisch verwaltet
DURATION	1 Wort	Vorzeichen stets positiv Darstellung in 1/1Ø sec : max 54 min
CLOCK	vorerst nicht implementiert	

2.2.2.2. Felder

Felder der Länge m (in CIMIC/1 ist nur diese Gesamtlänge bekannt) setzen sich aus m Einfachgrößen zusammen; bei String-Feldern bleiben dadurch u.U. einige Bytes bzw. Bits unbesetzt.

Vor dem 1. Feldelement wird als "Feldbeschreibungsvektor" die Anzahl der Feldelemente m als Integer abgesetzt; bei lokalen Feldern muß letzteres dynamisch erfolgen (bei Task- bzw. Prozedur-Eintritt); der Codegenerator schreibt entsprechende Anweisungen bei der Bearbeitung der Datendeklarationssequenz auf eine Hilfsdatei und kopiert diese bei Erscheinen der CIMIC/1-Anweisung "LINK" in den Ablauf-Code ein.

2.2.3. Registerbelegung

RØ: nicht belegbar (Rettregister für Anzeigen bei Ebenenwechsel)
R1: nicht belegbar (Rettregister für Befehlszähler)
R2: Indexregister (direkte Abb. des CIMIC-Indexreg.)
R3: "Freizeiger": Zeiger auf 1. freien Platz in den lokalen Daten
R4: "Lokalzeiger": Zeiger auf den jeweils aktuellen lokalen Bereich
R5: "Taskkopfzeiger": insbesondere Zeiger auf den Charakter-Akkumulator einer Task (R5 ist nach Initialisierung fest)
R6: Hilfsregister bei der Parameterübergabe
R7: Register für Rückkehradresse beim Unterprogr.-Sprung

GØ:
G1: } CIMIC-Akkumulator für FIXED, FLOAT, DURATION, BIT

G2:
G3: } Operand bei der Implementation einiger arithmetischer Befehle

G4: Hilfsregister für Referenzierung
G5: frei
G6: frei } (vorläufig)
G7: frei

} frei für Laufzeitprozeduren

Bemerkung:
G4 ist beim Aufruf einer LZP für diese prinzipiell frei; GØ bis G3 dienen zur schnellen Parameterübergabe an LZP-en und können von diesen nach Übernahme der Werte ebenfalls frei verwendet werden.

2.2.4. Verwendung von Dateien

Der obere Compilerteil legt den CIMIC/1-Code stets auf der Datei .CIM ab, das Programm CIZI erzeugt daraus einen Zifferncode, wobei der Dateiname wählbar ist. Der eigentliche Codegenerator CG1Ø liest diese Datei und erzeugt,damit die gewünschte Programmstruktur erreicht werden kann, mehrere Dateien (feste Namen), die durch das Programm CGBI dann in geeigneter Weise zu einer Datei zusammengefaßt werden.

Input-Dateien für CG1Ø:

XXXX	Cimic-Zifferncode
'TMC	Datei mit Macros für Taskkopf und Task-Initialisierungssequenz

Output-Dateien des CG1Ø:

'TUP	Moduldaten, Tasks u. Prozeduren
'TCB	Taskkontrollblöcke
'SCB	Semaphorkontrollblöcke
'EXT	Extern-Definitions-Anweisungen
'KAT	Aufruf-Struktur-Katalog (siehe dazu 2.2.5.)

Hilfsdateien des CG1Ø:

'QQQ	z. Zwischenspeichern von CIMIC/Code (lokale Hilfsvariable vor Parameterdeklaration
'QQF	z. Zwischenspeichern von Code für dynamische Vorbesetzung von Feldbeschreibungsvektoren lokaler Felder

Input-Dateien für CGBI:

....	alle Output-Dateien des CG1Ø, dazu als folgende Dateien mit festen Codestücken:
'OFS	Symbolische Offset-Definitionen der Taskkopf-Daten
'INI	Programm-Initialisierungsseqeunz
'DCB	Device-Kontrollblöcke
'LKT	Katalog der Laufzeitprozeduren

Steuerdateien des CGBI:

(enthalten Steuerbefehlssequenz für Standardläufe)

'HHH	Erzeugung eines Hauptmoduls auf Datei HHHH
'PPP	" " Prozedurmoduls auf Datei PPPP
'DDD	" " Stackmoduls auf Datei DDDD
HDHD	Zusammenfassung von 'HHH und 'DDD

Output-Dateien des CGBI:

abhgg. von Steueranweisung (s.o.)

Die Output-Dateien des Programms CGBI sind zugleich Input-Dateien für den Assembler ASSB, dessen Output-Datei-Name vom Benutzer wiederum frei gewählt werden kann.

Weitere Datei des Systems:

'SIG Informationsdatei für Benutzer, sie enthält eine numerische geordnete Liste aller Signals der Laufzeitprozeduren, mit Signalnummer, mnemonischem Text (der Ausgabe nach Systemreaktion) und Bedeutung des Signals.

2.2.5 Erzeugung eines Stackmoduls mit Optimierung des Platzbedarfs für lokale Daten

2.2.5.1. Problemstellung

Während der invariante Code von reentrant formulierten Prozeduren eines PEARL-Programms nur einmal im Speicher vorhanden zu sein braucht, müssen die lokalen variablen Daten der Prozeduren für jede aufrufende Task getrennt angelegt werden. Man kann sie als lokale Daten der Task in 2. Ordnung betrachten; das gleiche gilt natürlich auch für die lokalen Daten von Prozeduren aller weiteren Aufrufebenen. Die Referenzierung der lokalen Daten läßt sich sehr einfach implementieren, wenn man die lokalen Daten der verschiedenen Aufrufebenen hintereinander legt und Zeiger auf Anfang und Ende des lokalen Bereichs einführt (sogenannt:"Lokalzeiger" und "Freizeiger"), die bei Beginn und Ende einer Prozedur entsprechend weiter- bzw. zurückgesetzt werden.

Das Problem ist nun die Reservierung des gesamten Speicherplatzes für die lokalen Daten, die - wollte man sie ebenfalls dynamisch machen - einen kaum zu vertretenden Verwaltungsaufwand erforderte. Für die statische Reservierung durch Vorausberechnung ergeben sich Schwierigkeiten dadurch, daß (wegen

der Verwirklichung des Mehrmodulkonzepts) der Platzbedarf einer in einem anderen Modul liegenden Prozedur dem Codegenerator (zunächst) nicht bekannt ist.

2.2.5.2. Lösung

Die Berechnung muß in einem eigenen "Stackmodulerzeugungslauf" erfolgen. Hierzu wird bei der Übersetzung eines jeden Moduls vom Codegenerator (Progr. CG1Ø) eine sog. "Katalogdatei" erzeugt, die die benötigten Informationen über Aufrufstruktur und Eigenplatzbedarf der übersetzten Tasks und Prozeduren enthält. Die Katalogdateien aller zu einem Programm gehörenden Moduln und die Katalogdatei der Laufzeitprozeduren werden dann vom Programm CGBI eingelesen; dieses führt eine Aufrufstrukturanalyse durch, berechnet für jedes Objekt (Task oder Proz.) den für die lokalen Daten maximal benötigten Platzbedarf und erzeugt einen Assemblermodul, in dem für jede Task der entsprechende Bereich reserviert wird (→ Stackmodul). Der Beginn dieses Bereichs wird über einen generierten (vom Tasknamen abhängigen) globalen Namen dem Hauptmodul bekannt gemacht, so daß beim Start einer Task in der Taskinitialisierungssequenz Lokal- und Freizeiger entsprechend gesetzt werden können.

2.2.5.3. Details

a) Berechnung des Platzbedarfs

Der gesamte Platzbedarf an lokalen Daten eines Objekts berechnet sich aus dem Eigenplatzbedarf und dem größten Gesamtbedarf aller direkt von ihm aufgerufenen Prozeduren. Enthält die Aufrufstruktur direkte oder indirekte Rekursionen, so ist eine Berechnung daher nicht möglich. Rekursive Aufrufe sind daher in dieser Implementation nicht erlaubt.

b) Platz für Datenlisten

Die Datenlisten bei der formatierten E/A (siehe dazu 2.9.) sind als lokale Daten zu betrachten, für die im CIMIC/1-Code aber keine explizite Platzreservierungsanweisungen vorhanden sind. Da eine Datenliste nach Beendigung der E/A überschrieben werden kann, können sämtliche Datenlisten einer Task (auch die der von ihr aufgerufenen Prozeduren) übereinandergelegt werden. Die Berechnung des Maximalbedarfs für die Datenlisten und die Platzreservierung erfolgt ebenfalls durch das Programm CGBI; der CG setzt die hierzu nötige Information in den Katalogdateien ab.

c) Katalogdatei für Laufzeitprozeduren

Die Katalogdatei für die in Assembler geschriebenen LZ-Prozeduren wird von Hand erstellt. Aus diesem Grund haben die Kataloge auch nicht Binärform, vielmehr ist die Information leicht lesbar und editierbar in Textform dargestellt.

d) Syntax einer Katalogdatei

```
[objekt objektart :attribut LKB=lokdat   EAB=datlist ]°°°
[    prozname attribut]°°°
------
$$$$
```

Dabei ist:

objekt = Task- oder Prozedurname
objektart = PROC / TASK
attribut = G/M/L (Global, Modulebene, Laufzeitproz.)
lokdat = Zahl : Eigenbedarf an lokalen Daten in 16-Bit-Worten
datlist = Zahl: maximaler Eigenbedarf für Datenlisten
prozname = Name einer von 'objekt' aufgerufenen Prozedur

e) Fehlererkennung

Das Programm CGBI erkennt folgende Fehler :

- formale Fehler (syntaktisch falsch erstellter Katalog)
- Hauptmodulkatalog (mit Taskdeklaration) fehlt
- gleichnamige globale Prozeduren in verschiedenen Katalogen
- fehlende Prozeduren (bzw. fehlende Kataloge)
- Rekursionen

f) Erweiterungsmöglichkeiten

Die Kataloge können mit relativ geringem Aufwand erweitert werden. Sinnvoll wäre z.B. Information über Datentyp und Referenzstufe der Parameter, wodurch eine Überprüfung auf Gleichheit von Deklaration und Spezifikation in verschiedenen Moduln möglich wäre.

2.2.6. Spezifikationen zur Programmstruktur

CIMIC-Anweisung:	Erzeugter Code für 31Ø: (wenn nicht anders erwähnt, auf Datei 'TUP)

2.2.6.1. Start- und Finish-Anweisung (Beispiel)

START MOD1()	keine Codeablage auf Datei 'TUP. Der Initialisierungscode ist fest, steht auf der Datei 'INI un wird (beim Haptmodul) vom Programm CGBI eingebunden. Aber viele Initialisierungen im CG.
FINISH MOD1()	'NAME' MOD1 'SATZ' MOD1 } auf Datei 'EXT 'ENDE' auf Datei 'TUP Außerdem werden vom CG mehrere Aktionen abgeschlossen; insbesondere werden alle offenen Dateien geschlossen.

2.2.6.2. Systemteil

SPACE SDAU O DRUC() SPACE BSEA IO BILD()	Bei einem Hauptmodul werden stets für alle vom PBS verwalteten Geräte (der Standard-E/A und Analogeingabe) Device-Kontrollblöcke mit festen globalen Namen angelegt; der Code hierfür steht auf der Datei 'DCB und wird vom Programm CGBI eingebunden.
SPACE DED I DIGWIN() *1 SPACE DAS O DIGOUT() *5	} siehe Prozeß-E/A
SPACE SIGN(3) I OVERFL()	keine Code-Ablage, Platzreservierung im Taskkopf
SPACE ITRP(6) I DIGITR()	keine Code-Ablage (für die Verwaltung der Interrupts durch das PBS ist dort bereits Platz reserviert)

2.2.6.3. Moduldaten:(einige Beispiele)

```
SPACE INT VLMOD NØØØ1() A 17 >      'VF' NØØØ1/ = 17
SPACE REAL  MODUL NØØØ2() >         'VG' NØØØ2/ =
SPACE BIT(2) MODUL NØØØ3() >        'VF' NØØØ3/ =

SPACE CHAR(7) MODUL NØØØ4(5*CHAR(7))  >
          Feldlänge                 'VF'        = 5
      ein CHAR(7) belegt 4 Zellen   'VA' NØØØ4/ = 2Ø < >

SPACE REAL MODUL NØØØ5(1ØØ*REAL) >
                                    'VF'        = 1ØØ
                                    'VA' NØØØ5/ = 2ØØ < >

SPACE SEMA MODUL NØØØ6() A 1 >
                                      Anlegen eines Semaphorkontrollblocks
                                      auf Datei 'SCB  (mit Anfangswert 1) :
                                    'VA' NØØØ6/ = 3 < >
                                                = 1
```

2.2.6.4. Prozedurdeklaration: (Beispiel)

```
BEGIN PROC MODUL NØØØ7(3)  >
                                    keine Codeablage, aber natürlich viele
                                    Initialisierungsaktionen im CG (z.B.
                                    Namen merken, Anzahl der Parameter;
                                    lokalen Offset auf 22 setzen)
SPACE INT PARAM PØØØ1()  >          PØØØ1/'IDENT'22
                                    die Adressen aller lokalen Daten, wozu auch
                                    die Parameter gehören, werden in Offsets
                                    umgesetzt.
SPACE REAL VLPAR PØØØ2() >          PØØØ2/ 'IDENT'23
                                    Parameter per Wert (belegt, da
                                    REAL, 2 Zellen:  nächster Offset = 25)
SPACE BIT(3) PARAM PØØØ3(1Ø* BIT (3)) >
                                    PØØØ3/'IDENT'25
                                    Param. per Adresse belegt 1 Zelle
```

```
SPACE INT LOCAL NØØØ8()  >          NØØØ8/ 'IDENT'26
SPACE REAL LOCAL NØØØ9(1Ø*REAL) >
                                    NØØØ9/'IDENT' 28
                                    Hier wird (für die Feldlänge) zusätzlich
                                    ein Wort reserviert. Der CG muß sich
                                    merken, daß später Code für die dyna-
                                    mische Belegung der Feldlänge abzu-
                                    setzen ist.(siehe unten)
SPACE BIT(9) LOCAL NØØ1Ø()  >       NØØ1Ø/ 'IDENT' 48
SPACE INT VLLOC NØØ11() A 63 >
                                    NØØ11/  = 63
                                    Daten von Typ VLLOC werden wie VLMOD
                                    behandelt, also keine Offset-Definition
LINK PROC MODUL NØØØ7(3) >
                                    NØØØ7/
        Rücksprungadresse retten: 'VA'  (2Ø+R3):= R7   ┐
        alten 'Lokalzeiger retten:      (21+R3): = R4  ├ oder
        neuen'Lokalzeiger'setzen         R4    : = R3  ┘ G7 :US  REQLOK
        (auf den vorher freien Bereich)

        neuen 'Freizeiger' setzen,       R3    : = R3 + bedarf
        'bedarf' ist der für Parameter
        und lokale Daten benötigte Platz
        (für das obige Beispiel: 29)

        Einschreiben der        'VA'     GØ: = 1Ø
        Feldlänge  für das
        lokale Feld NØØØ9                (NØØØ9-1+R4) : = GØ
        (vgl. obiges Beispiel)
RETURN PROC MODUL NØØØ7(3) >

        Freizeiger restaurieren 'VA'     R3 : = R4      ┐
        Lokalzeiger restaur.             R4 : =(21+R3)  ├ oder
        Rücksprung                       :SP  (2Ø+R3)   ┘ : SP RELLOK

END PROC MODUL NØØØ7(3)          keine Codeablage, aber verschiedene
        CG-Aktionen, z.B. Aufrufliste der Katalogdatei abschließen.
```

2.2.6.5. Taskdeklaration (Beispiel)

```
BEGIN TASK MODUL TØØØ1(14)
      Anlegen eines Taskkontrollblocks        TØØØ1/ = 2
      auf Datei 'TCB gemäß 2.1.1.1.                  = 14
      14 ist Taskpriorität                           = 6
      TØØØ1B ist generierter Name für den            = TØØØ1B
      Beginn des ablauffähigen Code                  = 38 <>

      Anlegen eines Taskkopfs                 TØØØ1K/ = 21 <>
      (s. dazu 2.2.1)                                 :
      Hilfsmittel dazu ist die Datei                  :
      'TMC, die einen Macro hierfür
      enthält.
      TØØØ1K ist generierter Name
      für den Beginn des Taskkopf
      (zugleich Charakterakkumulator)

SPACE REAL LOCAL NØØ12()                    'VA' NØØ12/ 'IDENT'2Ø

      lokale Taskdaten werden (wie die von Prozeduren)
      als Offset deklariert, der hier aber mit 2Ø beginnt!

SPACE CHAR(5) VLLOC NØØ13() A TASKA

      VAL-Daten nicht als Offset;           'VA' NØØ13/='Z=TASKA '

LINK TASK MODUL TØØØ1(14)

      Erzeugen der Taskinitialisierungs-  TØØØ1B/  R5: = TØØØ1K
      sequenz(mit Hilfe von Macro auf             R4: = TØØØ1D-2Ø
      Datei 'TMC).                                R3: = R4 + 17
      Der Bedarf an lokalen Daten beträgt         (OFFSLZ+R5):=R4
      in diesem Beispiel 17 Worte                 (OFFSFZ+R5):=R3

END TASK MODUL TØØØ1(14)      keine Codeablage auf 'TUP, aber
                              mehrere CG-Abschlußaktionen, z.B.
                              Abschluß eines Katalogeintrags.
```

2.3. Referenzierung

2.3.1. Allgemeines:

Prozeduren, Tasks und Semaphore dürfen nur auf Modulebene deklariert werden (z.T. Einschränkungen dieser Implementation) und können daher direkt adressiert werden. Globale Objekte werden ebenfalls nur auf Modulebene deklariert und direkt adressiert, wobei bei der Deklaration auf der Datei 'EXT eine 'DEEX'-Anweisung, bei der Spezifikation eine 'NDEX'-Anweisung mit dem entsprechenden Namen abgelegt wird.

Ebenso können alle Modulgrößen und Konstanten (insbesondere auch die auf lokaler Ebene vom Typ VLLOC) direkt adressiert werden.

Lokale Größen, insbesondere per Wert übergebene Parameter, werden über den "Lokalzeiger" R4 mit dem CIMIC-Namen als Offset adressiert; diesem wird in der Datendeklaration per 'IDENT'-Anweisung ein entsprechender Wert zugeordnet.

Das CIMIC-Indexregister ist auf R2 abgebildet. Konstante CIMIC-Offsets werden vom CG in einen 310-Adreßoffset umgewandelt (in der folgenden Tabelle als CO bezeichnet).

Bei der Referenzierung von Feldelementen ist auch ein negativer CO möglich (der durch einen entsprechenden positiven Anteil im R2 ausgeglicher wird); der Assembler läßt zwar Offsetausdrücke mit negativen Termen zu, das Ergebnis der Ausdrücke darf allerdings nicht negativ sein. Deshalb kann die Zuordnung von lokalen Größen zu Offsets nicht mit dem Wert Ø beginnen, sondern erst mit 2Ø, da der maximale negative Offset -1*CHAR(4Ø) den Wert -2Ø hat.
Dies bedeutet aber gleichzeitig, daß der Lokalzeiger bei der Taskinitialisierung mit der um 2Ø erniedrigten Anfangsadresse des für die lokalen Daten reservierten Bereichs vorbesetzt wird.

Bei Parametern, die per Adresse übergeben werden, und bei indizierten lokalen Feldelementen (wobei R2 und R4 zugleich verwendet werden sollen), kann die Referenz zum Wert nicht mehr als ein einziger 'Adreßausdruck' des Assemblers notiert werden. In diesen Fällen erfolgt eine Adreßrechnung mit anschließender Referenzierung über ein Hilfsregister (G4).

2.3.2. Spezifikation (tabellarisch)

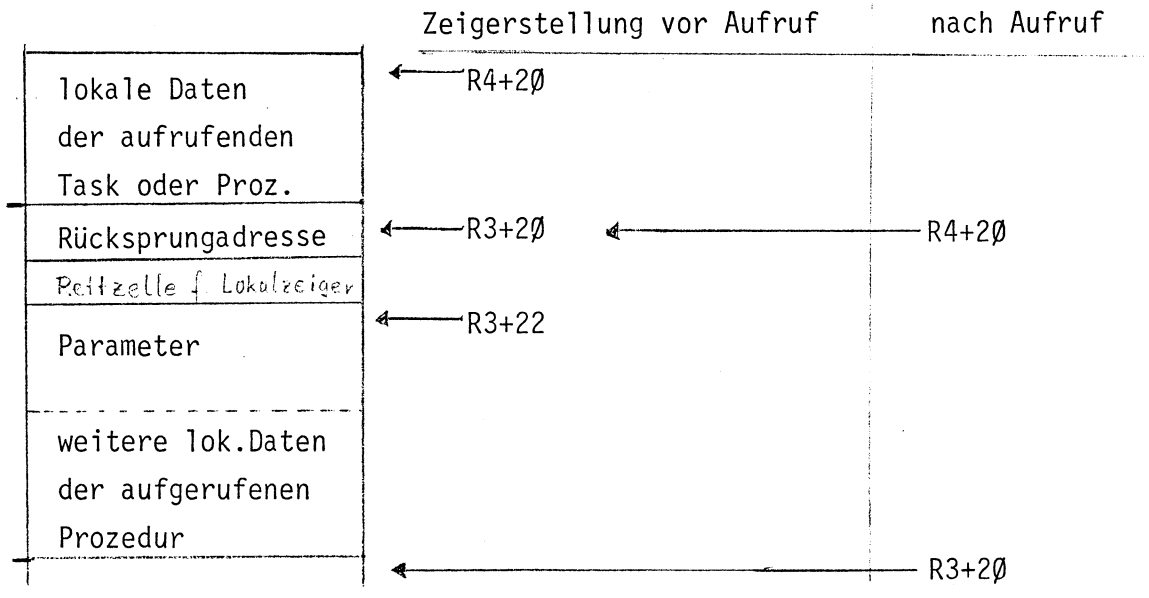
Kategorie	Objekt	Referenzierung: → Wert	Adresse
MODUL VLMOD	Einfachgröße	→(name)	name
CONST VLLOC	Feld	—	name
VLGLB GLOBAL CODE	Feldelement o.Indexr.	→(name+CO)	<name+CO>
	Feldelement m.Indexr.	→(name+CO+R2)	<name+CO> +R2
LOCAL ARG	Einfachgröße	→(name + R4)	name + R4
VLPAR	Feld	—	name + R4
	Feldelement o.Indexr.	→(name+CO+R4)	<name+CO> +R4
	Feldelement m.Indexr.	G4:= R2 + R4 →(name+CO+G4)	<name+CO>+R4+R2
PARAM	Einfachgröße	G4:=(name+R4) →(G4)	(name+R4)
	Feld	—	(name + R4)
	Feldelement o.Indexr.	G4:=(name+R4)+CO → (G4)	(name + R4)+ CO
	Feldelement m.Indexr.	G4:=(name+R4) +CO+R2 → (G4)	(name+R4)+CO+R2

2.3.3. Beispiele zur Referenzierung:

LOAD INT MODUL NØØØ1() >	'VF'	GØ:=(NØØØ1)
ADD INT MODUL NØØØ2(3*INT) >	'VF'	GØ:=GØ+(NØØØ2+3)
STORE REAL MODUL NØØØ3(-1*REAL)+ >	'VA'	(NØØØ3-2+R2) :=GØ (NØØØ3-2+1+R2) :=G1
LOAD,I CHAR(6) MODUL NØØØ4(5*CHAR(6)) >	'VA'	GØ:= NØØØ4+15
MPY REAL VLPAR NØØØ5() >	'VA'	G2:= (NØØØ5+R4) G3:= (NØØØ5+1+R4) R7:US REAMLT
LOAD BIT(16) LOCAL NØØØ6(-1*BIT(16))+ >	'VA'	G4:=R2+R4 GØ:=(NØØØ6-1+G4)
ADX INT LOCAL NØØØ7(4*INT) >	'VF'	R2:=R2+(NØØØ7+4+R4)
LOAD REAL PARAM NØØØ8() >	'VA'	G4:=(NØØØ8+R4) GØ:=(G4) G1:=(1+G4)
STORE REAL PARAM NØØØ9(-1*REAL)+ >	'VA'	G4:=(NØØØ9+R4)-2+R2 (G4):=GØ (1+G4):=G1
ARGIS,I INT PARAM NØØ1Ø ()* >	'VA'	GØ:=(NØØ1Ø+R4) (R6'):=GØ
LOAD CHAR(9) PARAM NØØ11(-1*CHAR(9))+ >		GØ:=(NØØ11+R4)-5+R2 G1:=9 R7:US STRLAD

2.4. Parameterübergabe

2.4.1. Lage des Parameterbereichs in den lokalen Daten



2.4.2. Prinzip der Parameterübergabe

Die Parameter werden vor dem Aufruf der Prozedur einzeln über das Register R6 mit Autoinkrement in den dafür vorgesehenen Bereich abgespeichert.
Bei Wertübergabe (ARGIS) wird der Wert des Arguments umgespeichert, wobei dieser zunächst in den 'CIMIC-Akku' geladen wird. Für Felder wird die Übergabe per Wert nicht zugelassen.
Bei Adreß-Übergabe (ARGIS, I und ARGIS,S) wird in jedem Fall die 'effektive' Adresse übergeben, d.h. z.B. bei lokalen Variablen die Summe von R4 und 'offset'.

2.4.3. Spezifikation (einige Beispiele; zum Verständnis ist die Kenntnis der Referenzierung günstig)

```
CALL PROC MODUL NØØ15(1Ø)   >          'VA'  R6: = R3 + 22
ARGIS INT MODUL NØØØ1(3*INT)  >              GØ: = (NØØØ1+3)
                                             (R6'):= GØ
ARGIS REAL MODUL NØØØ2()    >                GØ:= (NØØØ2)
                                             G1:= (NØØØ2+1)
                                             (R6'):= GØ
                                             (R6'):= G1
ARGIS BIT(8) LOCAL NØØØ3()  >                GØ:= (NØØØ3+R4)
                                             (R6'): = GØ
```

ARGIS CHAR(12) PARAM NØØØ4() >	GØ:=(NØØØ4+R4) G1:= 12 R7: US STRLAD
Übergabe mittels Laufzeitunterprogrammen zum Laden und Speichern des String-Akku; (Versorgung dieser Progr. mit Adresse und Länge!)	GØ := R6 G1:= 12 R7: US STRSTO R6: = R6 + 6
ARGIS, I DUR LOCAL NØØØ5() >	GØ:= NØØØ5+R4 (R6'):= GØ
ARGIS,I INT MODUL FELD()* >	GØ:= FELD (R6'):= GØ
ARGIS,I REAL LOCAL FELD2()* >	GØ:= FELD2+R4 (R6'):= GØ
ARGIS,I REAL LOCAL RØØØ3(3*REAL) >	GØ:=<RØØØ3+6> + R4 (R6'):= GØ
lokales Feldelement per Adresse	
ARGIS,I INT PARAM NØØØ6() >	GØ:= (NØØØ6+R4) (R6'):= GØ
ARGIS,S INT LOCAL RØØØ2() >	GØ:= (RØØØ2+R4) (R6'):= GØ
in RØØØ2 steht die eigentliche Adresse eines Feldelements	
INT LOCAL bezieht sich nicht auf RØØØ2, sondern auf das Datum, dessen Adresse in RØØØ2 abgelegt ist. Mode und Kategorie von RØØØ2 sind ADDR ARG	
CEND PROC MODUL NØØ15(1Ø) >	R7: US NØØ15

2.5. Tasking und Synchronisation

2.5.1. Tasking Operationen

Implementationsbedingt sind für die Tasking-Aufrufe folgende Schedules möglich:

	ACTIVATE	CONTINUE	RESUME	SUSPEND TERMINATE PREVENT
Ø = kein Schedule	x	x	-	x
5 = ALL	x	-	-	-
8 = AFTER	x	x	x	-
9 = AFTER ALL	x	-	-	-
12 = WHEN	x	x	x	-
13 = WHEN ALL	x	-	-	-

Die Tasking-Aufrufe werden vom PBS bearbeitet. Die dort verlangten Scheduleschlüssel stimmen mit denen im CIMIC-Code überein. Das in CIMIC durch den Leerstring dargestellte Fehlen eines Schedules wird an das PBS durch eine 'Ø' mitgeteilt. Der Aufruf an das PBS erfolgt - wie in 2.1.2. beschrieben - durch das Ablegen einer Auftrags-Parameterblockadresse in einer Kommunikationszelle und durch das Erhöhen eines Koordinierungszählers, was wiederum in der Laufzeitprozedur CALPBS erfolgt, die mit der Par.-Bl.-Adresse in GØ versorgt wird.

Das PBS überträgt die mit Schedule behafteten Auftragsblöcke in dafür vorgesehene Merkbereiche im Taskkontrollblock der Objekttask, so daß ein Auftragsparameterblock anschließend wieder für weitere Aufträge frei ist (Im Merkbereich der Objekttask kann natürlich immer nur ein Schedule stehen, d.h. daß nur der neueste Schedule Gültigkeit besitzt.

Für jede der sechs Taskoperationen wird pro Task im Taskkopf ein Parameterblock angelegt, in dem die Aufträge dieser Task an das PBS übergeben werden. Der P.Bl. ist statisch mit Funktionskennung, Zeiger auf den eigenen Taskkontrollblock und Leerschedulekennung vorbesetzt. Folgende Parameter müssen gegebenenfalls noch dynamisch eingetragen werden:

- aktueller Scheduletyp
- Interruptnummer
- Zeitdauer
- Zeiger auf Taskkontrollblock der Objekttask
- Schedulepriorität

Die Parameterblöcke können sowohl über einen generierten Namen als auch über den Taskkopfzeiger R5 mit festem Offset referenziert werden.

Beispiel:

ACTV TASK MODUL NØØ2Ø() >	GØ:=R5+OFFSAB G1:=NØØ2Ø (1+GØ):=G1
SCHED 13 >	G1:=13 (3+EØ):=G1
WHEN INTRP(6) I DIGITR() >	G1:=6 (8+GØ):=G1
ALL DUR CONST DØØØ1() A 15.ØSEC>	G1:=(DØØØ1) (1Ø+GØ):=G1
AEND >	R7 :US CALPBS
TERM TASK MODUL NØØ2Ø() >	GØ:=NØØ2ØT R7 :US CALPBS

(nur Eigen-Terminierung zugelassen)

2.5.2. Semaphor-Operationen

Die Semaphor-Operationen sind - wie die des Tasking - als Aufrufe an das PBS mit Übergabe eines Parameterblocks implementiert. Für REQUEST und RELEASE ist pro Task je ein Parameterblock (im Taskkopf) angelegt, der mit Funktions- und Leerschedule-Kennung statisch vorbesetzt ist. Die Verweisadresse auf das aktuelle Semaphore wird dynamisch eingetragen. Nach der Bearbeitung des Aufrufs durch das PBS ist der Parameterblock wieder frei für den nächsten Eintrag.

Beispiel:

REQU SEMA MODUL NØØ22() >	GØ:=R5+OFFSRQ G1:=NØØ22 (1+GØ):=G1 R7 :US CALPBS
RELE SEMA GLOBAL PUFVOL() >	GØ:=R5+OFFSRL G1:=PUFVOL (1+GØ):=G1 R7 :US CALPBS

2.6. Signalbearbeitung

Die Siganleinplanung und -Reaktion erfolgt durch zwei Routinen des Laufzeitpakets. Dazu werden am Ende des Taskkopfs für jedes im Systemteil deklarierte Signal zwei Speicherworte reserviert (unabhängig davon, ob das Signal in der jeweiligen Task auch eingeplant wird). Diese Signalliste wird mit Nullen vorbelegt und erhält ein Endekennzeichen (-1).

Die Einplanung zur Laufzeit erfolgt durch die LZ-Prozedur SGNEIN, der als Parameter die Signalnummer und die Adresse des Sprungziels bzw. Ø bei SYSTEM-Reaktion übergeben werden. Da die Sprungzieladresse also bereits bei der Einplanung bekannt sein muß, werden bei der vorliegenden Implementation indizierte Sprünge über Labelarrays bei Signaleinplanungen nicht zugelassen. Die Prozedur SGNEIN trägt die Parameter in die Signalliste ein.

Die Erzeugung des Signals erfolgt in den einzelnen Laufzeitprozeduren. Dabei wird die LZ-Prozedur SIGREA aufgerufen, der als Parameter die Signalnummer und ein Mnemocode übergeben werden. Die Prozedur SIGREA prüft, ob die Nummer in der Signalliste eingetragen ist. Wenn nicht, oder wenn explizit die Systemreaktion eingeplant ist, wird eben diese ausgeführt: Bei der vorliegenden Implementation besteht sie darin, daß auf dem Bedienungsgerät (Bildschirm) der Taskname (← Taskkopf) und der Mnemocode ausgegeben und die Task terminiert wird.

Liegt dagegen eine "positive" Einplanung vor, so wird zu der angegebenen Marke gesprungen. Zuvor müssen allerdings Lokal- und Freizeiger auf die Werte gesetzt werden, die sie in der Ebene der Reaktionsmarke haben. Um den Implementationsaufwand hierfür gering zu halten, sind deshalb Einplanungen nur in Tasks zugelassen, wo die Zeiger eindeutige (bei der Taskinitialisierung gesetzte und gerettete) Werte haben.

Spezifikationen:

```
SPACE SIGN(17) I OVERFL()  >
     Systemteilanweisung: keine Codeablage, lediglich die Gesamt-
                          zahl der deklarierten Signals wird erfaßt
                          (für die Länge der Signalliste im Taskkopf)

ONC SIGN(17) I OVERFL()  >                    GØ:=7
ONL,I INSTR CODE MØØØ9() >                    G1:=MØØØ9
                                              R7 :US SGNEIN

SYSTEM     >                                  G1:=Ø
                                              R7 :US SGNEIN
```

2.7. Algorithmische Funktionen

Die algorithmischen Funktionen werden teils durch direkte Codeablage (v.a. bei den Datenarten INT, DUR und BIT) teils durch Laufzeitprozeduraufrufe mit geeigneter Parameterversorgung (v.a. bei REAL und CHAR) implementiert. Im Gegensatz zu Anwenderprozeduren werden die Parameter ausnahmslos in Registern übergeben.

2.7.1 Arithmetische Funktionen

Die folgenden Spezifikationen sind nach der Datenart aufgegliedert. Für die in Kap. 2.3. genau beschriebene Referenzierung wird dabei meist die abkürzende Bezeichnung 'ref' verwendet.

2.7.1.1 Integer-Größen

LOAD		G0:= ref	
STORE		ref:= G0	
STN		wie STORE	
LDX		R2:= ref	
STX		ref:= R2	
NEG		G1:= G0	
	'VF'	GO:= 0-G1	
ABS	'VF'	G0<0 : SP 'AP'+4	
		G1:= G0	
		G0:= 0-G1	
ADD	'VF'	G0:= G0+ref	
SUB	'VF'	G0:= G0-ref	
MPY		G2:= ref	
		R7: US INTMLT	[G0:= G0*G2]
DIV		G2:= ref	
		R7: US INTDIV	[G0:= G0/G2]
EXP		G2:= ref	
		R7 : US INTEXP	[G0:= G0**G2]
ADX	'VF'	R2:= R2+ref	
SBX	'VF'	R2:= R2-ref	
MPX		G2:=ref	
		R7 : US INXMLT	[R2:= R2*G2]

ADDAX 'VF' R2:= R2 + GØ

CMP Der Vergleichsbefehl wird mit dem hierauffolgenden bedingten Sprungbefehl in einer Assembleranweisung abgesetzt:

'VF' GØ vglop ref : SP reference-5

für vglop sind, den Sprungbedingungen entsprechend, folgende Zeichenfolgen einzusetzen:

JEQ =
JNE #
JGE >=
JGT >
JLE <=
JLT <

2.7.2.2. Real-Größen

Von Laden und Speichern ausgenommen, werden alle Operationen durch Laufzeitprozeduren ausgeführt, die den 1. Operanden in GØ und G1, einen zweiten - wenn benötigt - in G2 und G3 übernehmen und das Ergebnis wieder in Gø und G1 abliefern.

Die im folgenden verwendete Bezeichnung 'ref+' bedeutet, daß durch einen an geeigneter Stelle des Referenzausdrucks eingefügten Offset von '+1' das zweite Wort einer REAL-Größe angesprochen wird.

LOAD 'VA' GØ:= ref
G1:= ref+ d.h. mit Zusatzoffset +1

STORE 'VA' ref:= GØ
ref+:= G1 s.o.

STN wie STORE

NEG R7 : US REANEG

ABS R7 : US REAABS

ADD G2:= ref
G3:= ref+
R7 : US REAABS

SUB G2:= ref
G3:= ref+
R7 : US REASUB

MPY :
R7 : US REAMLT

DIV
:
R7 : US READIV

EXP
:
R7 : US REAEXP

CMP
G2: = ref
G3: = ref+
R7: US REACMP
Die Prozedur REACMP setzt das PZR-Register so, daß im folgenden bedingten Sprungbefehl die in 2.7.1.1. genannten Vergleichsoperatoren gesetzt werden können:
vglop : SP reference - 5

2.7.1.3. Zeitdauergrößen

Die Operationen stellen eine Untermenge der Integeroperationen dar. Die stets positiven DUR-Größen wurden daher ganz analog behandelt.

2.7.1.4. Zeichenketten

Zeichenkettenoperationen werden ausnahmslos durch Laufzeitprozeduren implementiert. Diese verarbeiten den Stringakku, der über R5 referenziert wird, mit einem Operanden, dessen Adresse in GØ und dessen Länge in G1 übergeben wird. Das Ergebnis wird (samt Länge) im Stringakku zurückgegeben (ausgen. Speicherbefehl).

LOAD	GØ: = ref	Quell-Ref.
	G1: = länge	
	R7: US STRLAD	
STORE	GØ: = ref	Ziel-Ref.
	G1: = länge	
	R7: US STRSTO	

Die Prozedur führt auch die Umwandung CHAR(i) → CHAR (j) durch , wenn die Länge im Stringakku nicht mit der in G1 übereinstimmt.

CONC	GØ: = ref	Quell-Ref.
	G1: = länge	
	R/: = US STRCON	
CHARAC	G2: = GØ	Nummer des zu selekt. Zeichens
	GØ: = ref	Quell-Ref.
	G1: = länge	
	R7: US STRSEL	

```
CMP         GØ: = ref
            G1: = länge
            R7: US STRCMP
```

Die Prozedur STRCMP setzt das PZR*) so, daß nachfolgend ein 'indirekter Vergleich' mit den Operatoren '#' und '=' abgesetzt werden kann.

2.7.1.5. Bitketten

Bitketten sind linksbündig in einem Wort dargestellt. Da bei den CIMIC-Anweisungen stets die Länge der Bitketten angegeben wird, braucht diese nicht prinzipiell im abgelegten Code mitgeführt zu werden; der Codegenerator merkt sich die aktuelle Länge des CIMIC-Akku und setzt sie bei Bedarf ein.

```
LOAD        GØ: = ref
STORE       ref: = GØ
NOT     'VA'GØ:  = GØ . X 'M = 1111100000000000'
                              soviele '1'en , wie Länge
AND         GØ: = GØ . U  ref
OR          GØ: = GØ . O  ref
EXOR        GØ: = GØ . X  ref
CONC        G2: = ref
            G2: = G2 . V  länge (Länge von Akku!)
            GØ: = GØ . O  G2    in CG neue Länge errechnen

LBIT        G2:= ref -1           ref=Nummer des zu selektierenden Bits
            R7 :US SHIFTE         Bitmuster nach links schieben
            GØ: =GØ .U 'H=8ØØØ'   Ergebnis ausblenden

RBIT        G2: = länge - ref     wie oben
            R7 :US SHIFTE
            GØ: =GØ .U 'H=8ØØØ'

SHIFT       G2: =ref
            R7 : US SHIFTE
            GØ: = GØ.U maske (wie bei NOT-Befehl)

CMP         wie bei Integer; Vergleichsoperatoren aber nur '=' und '#'
```

*) Programm-Zustands-Register der 31Ø

2.7.2. Konversionsfunktionen

Die Datentypkonversionen bei der CIMIC/1-Anweisung STORE werden teils als Code, teils durch Aufrufe (bzw. Aufruffolge) von entsprechende Laufzeitprozeduren implementiert:

INT → REAL		R7 :US CONVIR
INT → BIT		G1:=länge R7 :US CONVIB
REAL → INT		R7 :US CONVRI
REAL → BIT		R7 :US CONVRI G1: =länge R7 :US CONVIB
BIT → INT	'VF'	GØ: = GØ .V <länge -16>
BIT → REAL	'VF'	GØ: = GØ .V <länge -16> R7 : US CONVIR
CHAR(i)→CHAR(j)		in STRSTO enthalten

2.8. Prozeß - E/A

Die Prozeß-E/A umfaßt den Betrieb der Digital-Ein-/Ausgabe sowie der Analog-Eingabe mit MOVE-Operationen. Die Digital-Ein-/Ausgabe (von BIT-Größen) erfolgt ohne OPTIONs. Bei der Analog-Eingabe (von INT-Größen) wird mit Hilfe von Options der Meßbereich und Steuerparameter angegeben. Bei Eingabe von Analogwerten von einem Gerätefeld in ein Internfeld kann über Steuerparameter zwischen einem sequentiellen Eingabe-Modus und einem "quasi-parallelen" Eingabe-Modus unterschieden werden.

Der Anschluß der Prozeß-E/A-Geräte im Systemteil entspricht nicht ganz dem allgemeinen Übereinkommen: So bedeutet die Anschluß-Nummer bei der Digital-E/A den Steckplatz im ZE-Rahmen, bei der Analog-Eingabe dagegen die Meßstellen-Nummer. Der Grund hierfür liegt in dem eingeschränkten Sprachvorrat des vorhandenen Systemteilübersetzers (er erlaubt Anschlußbildung nicht direkt an der ZE) und der unterschiedlichen System-Behandlung von Digital-E/A (nur über Bit-Befehle) und Analog-Eingabe (nur über BS-Aufrufe). Zur Ausnutzung der Geräteeigenschaften der (integrierenden) Analog-Eingabe ist die Bildung von Gerätefeldern vorgesehen.

CIMIC	ASSEMBLER
SPACE *DED* I dref()*anschluß >	dref/ anschluß | *D* | *E*
SPACE *DAS* 0 dref()*anschluß >	dref/ anschluß | *D* | *A*
SPACE *ANE* I dref()*anschluß >	dref/ anschluß — anschluß< 256
SPACE *ANE* I dref(dleng)+ >	dref/ dleng — dleng< 256, anschluß< 256
SPACE *ANE* I(1) *anschluß$_1$ >	anschluß$_1$ — dleng
≀	
SPACE *ANE* I(1) *anschluß$_{dleng}$ >	anschluß$_{dleng}$ — dleng

CIMIC | ASSEMBLER

DIGITALEINGABE

IMOVE DED I dref() > 'dref' einzelgerät

OPT >

UREAD BIT(16) LOCAL iref

LDX INT LOCAL N....() > MPX INT CONST IE216() E BIT (16) > (-1*BIT(16))+ >	'iref'=feldelement dynamisch
(1*BIT(16)) >	'iref'=feldelement statisch≠1
() >	'iref'=feldelement statisch=1 oder einzelgröße

(dref merken)

Codesequenz zur Berechnung von → iref

GØ:= dref
G1:= iref
R7 :US DIGEIN

DIGITALAUSGABE

LDX INT LOCAL N....() > MPX INT CONST IE 216() E BIT(16) > LOAD BIT (16) LOCAL iref (-1*BIT(16))+ > STORE BIT (16) LOCAL B....() >	'iref'=feldelement dynamisch
LOAD BIT(16) LOCAL iref ([1*BIT(16)]) > STORE BIT(16) LOCAL B....() >	'iref'=feldelement statisch≠1 []gilt bei 1
/	'iref' = einzelgröße

→ B....

OMOVE DAS O dref() > 'dref'einzelgerät

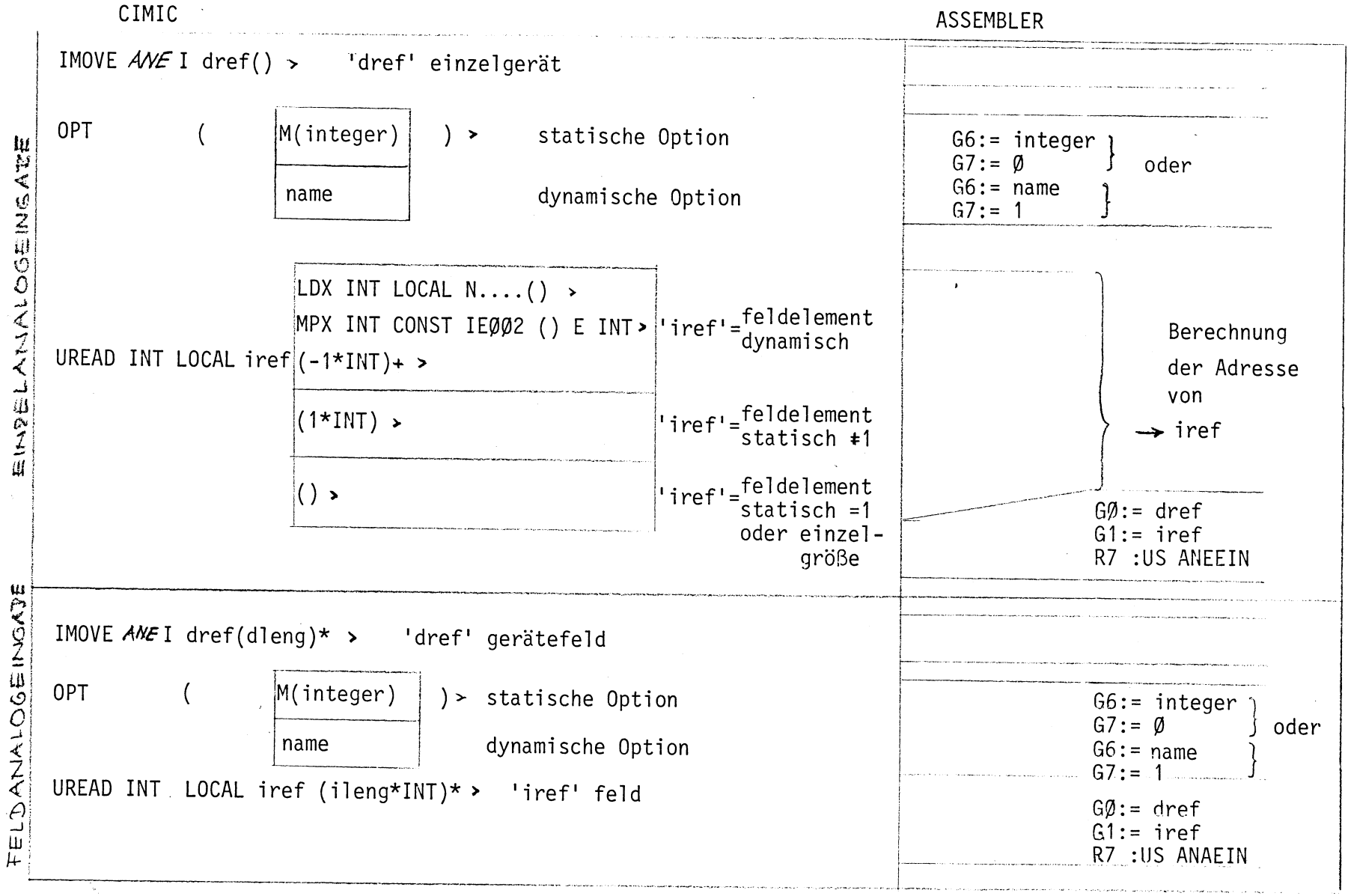
OPT >

UWRITE BIT(16) LOCAL

B....() >	'iref'=feldelement
iref() >	'iref'=einzelgröße

(dref merken)

GØ:= dref
G1:= iref oder B....
R7: US DIGAUS


CIMIC
ASSEMBLER
EINZELANALOGEINGABE
IMOVE ANE I dref() > 'dref' einzelgerät
OPT (M(integer)) > statische Option
name dynamische Option
G6:= integer
G7:= Ø
G6:= name
G7:= 1
oder
UREAD INT LOCAL iref
LDX INT LOCAL N....() >
MPX INT CONST IEØØ2 () E INT >
(-1*INT)+ >
'iref'=feldelement dynamisch
(1*INT) > 'iref'=feldelement statisch ≠1
() > 'iref'=feldelement statisch =1 oder einzelgröße
Berechnung der Adresse von → iref
GØ:= dref
G1:= iref
R7 :US ANEEIN
FELDANALOGEINGABE
IMOVE ANE I dref(dleng)* > 'dref' gerätefeld
OPT (M(integer)) > statische Option
name dynamische Option
UREAD INT LOCAL iref (ileng*INT)* > 'iref' feld
G6:= integer
G7:= Ø
G6:= name
G7:= 1
oder
GØ:= dref
G1:= iref
R7 :US ANAEIN

2.8.4. Laufzeitfunktionen der Prozeß-E/A (Funktionsübersicht)

DIGEIN /lokalen Datenbereich sichern
Lokalzeiger einstellen

Anschluß feststellen

Bitmuster eingeben
Anzeigen auswerten
Bitmuster abspeichern
lokalen Datenbereich freigeben

DIGAUS /lokalen Datenbereich sichern
Lokalzeiger einstellen

Anschluß feststellen
Bitmuster umspeichern
Bitmuster ausgeben
Anzeigen auswerten
lokalen Datenbereich freigeben

ANAEIN /lokalen Datenbereich sichern
Lokalzeiger einstellen

Parameter überprüfen
Meßstellenanzahl eintragen
Referenz auf Meßstellenliste eintragen

Options eintragen, ggf. mit Typ-Prüfung

PBS-Aufruf anstoßen

Warten
Anzeigen auswerten
Meßwert(e) umspeichern
lokalen Datenbereich freigeben

2.9. Standard-E/A

2.9.1. Prinzip

Für die Standard-E/A werden folgende Kontroll- bzw. Datenstrukturen verwendet:

a) Formatliste
b) Datenliste
c) Zeichenpuffer
d) Auftragsparameterblock
e) Device-Kontrollblock

Da nicht nur die Formatierung einzelner Daten, sondern auch die Abarbeitung der Formatliste durch reentrant übersetzte PEARL-Prozeduren erfolgt, hat es sich als sinnvoll erwiesen, die Auflösung der (in einem Quellprogramm vorhandenen) Datenliste in einzelne E/A-Aufrufe (in CIMIC/1) wieder rückgängig zu machen, d.h. eine Datenliste anzulegen. An die PEARL-Laufzeitprozedur EASTND werden daher folgende Parameter (z.T. als Felder vom Typ FIXED) übergeben:

Formatliste, Datenliste, Auftragsparameterblock, Datentransferrichtung. Die Übergabe dieser Parameter wiederum erfolgt aus Platzersparnisgründen durch die in Assembler geschriebene Laufzeitprozedur STNDEA, die zudem die Adresse des Devicekontrollblocks in den Auftragsparameterblock einträgt.

Die E/A erfolgt zeilenweise: bei der Ausgabe wird durch das E/A-Paket in einer Laufschleife jeweils eine Zeile erzeugt (d.h. ein Zeichenpuffe gefüllt) und über PBS-Aufrufe ausgegeben, bis die Datenliste abgearbeitet ist; bei der Eingabe wird jeweils über PBS-Aufruf eine Zeile gelesen und umgewandelt.

2.9.2. Erzeugung der Datenstrukturen

2.9.2.1. Formatliste

Die Formatliste wird in CIMIC/1 als Feld von Formatelementen wie die Deklaration eines Konstantenfelds angelegt, innerhalb einer E/A-Anweisungssequenz wird nur noch die Referenz auf dieses Feld verwende Die Formatlistenelemente sind vom Typ VLMOD bzw. VLLOC und können daher vom CG statisch angelegt werden. Für jedes Element werden dazu zwei Worte benötigt.

Allgemeine Form eines Elements in CIMIC/1:

[(a)] f [(w [,d])]

a = Wiederholungsfaktor

f = Formattyp

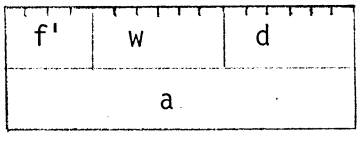
w = Feldlänge

d = Dezimalstellenanzahl.

Der Formattyp wird gemäß folgender Tabelle als Zahl f' verschlüsselt:

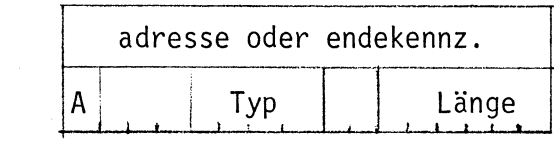
f	A	B1	B3	T	F	E	D	SKIP	PAGE	X	(	)
f'	1	2	3	4	5/11	12	13	6	7	8	9	1Ø

Dabei wird F verschieden verschlüsselt, je nachdem ob es sich um den Typ FIXED (=5) oder FLOAT (=11) handelt, was am Vorhandensein der Dezimalstellenanzahl festgestellt wird. Die Zahlen a,f', w und d werden vom CG gemäß der folgenden Darstellung in zwei Worte gepackt, wobei für nicht vorhandene Zahlen Ø eingesetzt wird.



2.9.2.2. Datenliste

Die Datenliste wird dynamisch in einem extra hierfür reservierten Bereich angelegt. Der pro Task benötigte Platz wird ähnlich wie der für die lokalen Daten berechnet und optimiert (siehe dazu auch 2.2.5.3.). Aus jeder Cimic-Anweisung CREAD bzw. CWRITE innerhalb einer GET- bzw. PUT-Anweisung erzeugt der CG Code, durch den ein entsprechendes Datenlistenelement in die Datenliste eingeschrieben wird. Die Datenliste wird durch eine Endekennung (-1) abgeschlossen.
Jedes Element der Datenliste besteht aus Adresse des Datums, Datentyp, Länge (bei Zeichen- und Bitketten) und Arraykennung. Diese Angaben werden folgendermaßen (teils verschlüsselt) in zwei 16-Bit Worte gepackt:



Länge d.h. Anzahl der Zeichen 1...4Ø
" " Bits 1...16

array
kennung:

Ø = Einfachgröße
1 = Feld

Datentyp	Schlüssel
FIXED	Ø
FLOAT	1
BIT	2
CHAR	3
DUR	4
CLOCK	5

2.9.2.3. Zeichenpuffer

Der Zeichenpuffer ist ein lokales Feld der Prozedur EASTND. Die Länge von 42 Worten setzt sich zusammen aus:

1. Wort = Pufferendeadresse (siehe dazu 2.1.2.2.)
2.-41. = 8Ø Zeichen
42. = Abschlußzeichen ETX

2.9.2.4. Auftragsparameterblock

Er wird im Taskkopf angelegt (mit statischer Vorbesetzung von Funktions- und Leerschedulkennung). Zur Laufzeit wird (von der Proz. STNDEA) die Adresse des Decive-Controllblocks und (vom E/A-Paket) die Adresse des Zeichenpuffers eingetragen.

2.9.2.5. Device-Kontrollblock

Für die Geräte der Standard-E/A werden (im Hauptmodul) die Kontrollblöcke prinzipiell und mit festgelegtem Namen angelegt, und zwar je einer für Eingabe von Konsole (BSEAII), Ausgabe auf Bildschirm (BSEAOO) und Ausgabe auf Schnelldrucker (SDAUOO).

2.9.3. Spezifikation (Beispiel)

```
PUT SDAU 0 DRUCK()  >          Name 'SDAU' merken
                               R6:= (OFFSDL+R5)
                                   Zeiger auf Datenliste setzen

FORMAT FORM VLLOC NØØ7Ø() >    Formatlistennamen NØØ7Ø merken

CWRITE REAL LOCAL NØØØ4() >    GØ:= NØØØ4 + R4
                               (R6'):= GØ
                               Adresse des Datums in Datenliste eintragen
                               GØ:='H=Ø1ØØ'
                               (R6'):= GØ
                               Datenbeschreibung eintragen

PEND     >                     GØ:= -1    Endekennung
                               (R6):= GØ

                               GØ:= SDAUOO    Gerät
                               G1:= NØØ7Ø     Formatliste
                               G2:= 1         Übertragungs-
                               R7 : US STNDEA  richtung
```

3. Laufzeitprozeduren

3.1. Allgemeines

Die Laufzeitprozeduren untergliedern sich in Standardfunktionen (bzw. -Prozeduren) und Systemprozeduren. Die Standardfunktionen werden vom Anwender wie PEARL-Prozeduren aufgerufen, die in einem anderen Modul deklariert sind. Die explizite Spezifikation kann dabei unterbleiben, wenn sie dem oberen Compilerteil durch Eintrag in die Liste der Standardfunktionen bekannt gemacht ist.
Aufrufe von Systemprozeduren werden vom Code-Generator erzeugt. Bei der vorliegenden Implementation werden die Parameter im Unterschied zu den Standardfunktionen zwecks Platz- und Laufzeitoptimierung grundsätzlich in Registern (GØ bis G7) übergeben.

Die Laufzeitprozeduren sind prinzipiell reentrant. Eine Ausnahme stellen lediglich die Filehandlingprozeduren und die Prozeduren zur E/A auf den Kopplungsbaustein dar (Die Reentrantfähigkeit dieser Prozeduren könnte erreicht werden, wenn man die in ihnen vorhandenen ORG-Aufrufe durch das PBS verwalten lassen würde, was aber den Anwender nicht von gewissen Koordinierungsmaßnahmen befreien würde).

3.2. Kurzbeschreibung der implementierten Standardfunktionen

3.2.1. Typwandlungsfunktionen

FIX	wandelt	FLOAT-Wert	in FIXED-Wert
FLOAT	wandelt	FIXED-Wert	in FLOAT-Wert
SYMBOL	wandelt	FIXED-Wert	in CHAR(1)-Wert
CBIT	wandelt	CHAR(1)Wert	in BIT(8)-Wert

3.2.2. Arithmetische Funktionen

ABS ergbit Absolutbetrag einer FLOAT-Größe
SQRT Quadratwurzel einer FLOAT-Größe

3.2.3. Ein/Ausgabe von Zeichenfeldern von/auf Koppelstrecke

TTY1IN	Eingabe
TTY1OU	Ausgabe

Als Parameter werden ein CHAR(2)-Feld und die Anzahl der zu übertragenden Zeichenpaare übergeben.

3.2.4. File-Handling-Prozeduren für dateiorientierte E/A auf Floppy-Disk

Leistungsumfang:
-sequentielles Lesen und Schreiben von Daten im Binärformat
- Datentypen: FIXED, FLOAT, CHAR(40)
- Datenanzahl pro Transfer im Aufruf wählbar
- Sicherheit durch Prüfung von Parametern und Aufruffunktionen (u.a. anhand eines Statuskontrollworts in einem File-Kontrollblock)
- Signals bei Programmier- und Hardwarefehlern (insbes. beim Eröffnen einer nicht vorhandenen Datei für Eingabe und einer bereits vorhandenen Datei für Ausgabe)

Prozedurnamen	und Bedeutung:
FILOPE	Eröffnen einer Datei für Input
FILCRE	Eröffnen einer Datei für Output
FILCLO	Schließen einer Datei
FILDEL	Löschen einer Datei
FFLINP FFIINP FCHINP	Einlesen von einer Datei in ein Feld vom Typ FLOAT, bzw. FIXED bzw. CHAR(40)

FFLOUT FFIOUT FCHOUT	}	Ausgabe von einem Feld auf eine Datei

3.3. Besonderheiten der Systemprozeduren

- Die Prozeduren zur formatierten E/A sind in PEARL portabel formuliert (siehe dazu /9/). Der Implementationsaufwand steht im umgekehrten Verhältnis zu Länge und Laufzeit des E/A-Pakets.
- Die Prozeduren des Stringhandlings wurden ebenfalls portabel formuliert /13/ und gemäß den Spezifikationen eines Umsetzprogramms per Hand in Assembler übersetzt. Der so entstandene Code kann sowohl hinsichtlich Platzbedarf als auch in Bezug auf die Laufzeit als fast optimal bezeichnet werden.
- Für die arithmetischen Operationen bei FLOAT-Größen wurde die Simulation durch das ORG31Ø /12/ verwendet.

4. Literaturverzeichnis

/1/ P. Holleczek et al.: PEARL Cross-Compiliersystem für den Prozeßrechner Siemens 31Ø; Regelungstechnische Praxis 1979, Heft 12

/2/ P. Holleczek: Erfahrungen mit portabler Systemsoftware bei PEARL-Implementationen für die Prozeßrechner Siemens 3Ø6 und 31Ø; Portable Software, Tagung I/1980 des German Chapter of ACM, H.J. Schneider (Hrsg.), B.G. Teubner, Stuttgart, 1980

/3/ P. Elzer, P. Holleczek: ASME-Compiler wird der Öffentlichkeit vorgestellt. Regelungstechnik 12 (1975)

/4/ B.Eichenauer: Entwicklungskonzept für ein Programmsystem zur Erstellung und Erzeugung von Systemsoftware für Prozeßautomation; PDV-Mitteilungen 1/1971; Ges. f. Kernforschung mbH, Karlsruhe

/5/ Mühlhahn: Spezifikation CIMIC/1, Bericht KFK-PDV 75, Ges. f. Kernforschung mbH, Karlsruhe 1976.

/6/ W. Eberlein: Implementation eines Zwischensprache-Transformators, Diplomarbeit am IMMD II, Universität Erlangen 1979.

/7/ R. Rössler: Betriebssystemstrategien zur Bewältigung von Zeitproblemen in der Prozeßautomatisierung; Disseration, Universität Stuttgart 1979.

/8/ A. Fleischmann, F.-J. Prester: Ein PEARL-Betriebssystem für die 31Ø. PDV-Entwicklungsnotiz, Kernforschungszentrum Karlsruhe, GmbH - wird veröffentlicht

/9/ K. Lampe, F.J. Prester: Anpassung der rechnerunabhängigen Laufzeitroutinen der PEARL-Standard-E/A für die Siemens 31Ø. PDV-Entwicklungsnotiz, Kernforschungszentrum Karlsruhe GmbH, wird veröffentlicht.

/1Ø/ Gruber, Inderst, Piche: Programmieranleitung für das AME-1-PEARL-SUBSET; Bericht KFK-PDV 1ØØ, Ges. f. Kernforschung mbH, Karlsruhe 1976

/11/ Programmiersprache PEARL, Basic-PEARL, Entwurf DIN 66253, 1978

/12/ SIEMENS SYSTEM-3ØØ Organisationsprogramm ORG31Ø/MOBI. Beschreibung P71ØØ-BØ31Ø-x-A3-35

/13/ W. Lindstedt: Portabilität algorithmischer Laufzeitprogramme, Portable Software, Tagung I/1980 des German Chapter of ACM, H.J. Schneider (Hrsg.), B.G. Teubner, Stuttgart 1980