PDV

Projekt Prozeßlenkung mit DV-Anlagen

Entwicklungsnotizen

PDV-E 88

CIMIC/C (PEARL-Compiler-Set)

GPP, Gesellschaft für Prozeßrechner-
programmierung mbH, München

Oktober 1976

GESELLSCHAFT FÜR KERNFORSCHUNG

PDV-E 88

CIMIC/C (PEARL-Compiler-Set)

GPP, Gesellschaft für Prozeßrechner-
programmierung mbH, München

Oktober 1976

Spezifikation der Zwischensprache

C I M I C / C

(PEARL - Compiler - Set)

Muenchen, den 27.9.1976

Verfasser: B. F. Eichenauer

G P P

Gesellschaft fuer Prozessrechnerprogrammierung mbH

Balanstrasse 138/I

8000 Muenchen 90

GPP/15/76

Zusammenfassung

Es wird die Untermenge CIMIC/C der Zwischensprache CIMIC/P beschrieben, in welcher der portable GPP-PEARL-Compiler fuer den PEARL-BASIS-SUBSET dargestellt wird.

Inhalt

1. Einfuehrung

Eine wesentliche Aufgabe, die bei der Erstellung von Basis-Software fuer Prozessrechner heute geloest werden muss, besteht in der Bereitstellung problemgemessener Programmier-Werkzeuge, mit denen sich einerseits der durch die Personalleistungen anfallende zunehmende Kostendruck wenigstens teilweise auffangen laesst, und die es andererseits ermoeglichen, ganz oder in Teilen wiederverwendbare Programmpakete fuer die Prozessautomatisierung zu erstellen.

Mit der Definition solcher Programmier-Werkzeuge, zu denen insbesondere die Prozess-Programmiersprache PEARL gehoert, ist die angegebene Aufgabe jedoch noch nicht geloest. Beruecksichtigt man die Vielzahl der heute auf dem Markt vorhandenen und erst recht der in naher Zukunft zu erwartenden billigen Rechensysteme, so ist die Spezifikation solcher Programmier-Werkzeuge nur sinnvoll, wenn diese sich schnell und mit angemessenen Kosten bereitstellen lassen, d.h. bei der Erstellung von Prozess-Automatisierungsprogrammen tatsaechlich einsetzbar sind.

Mit der Spezifikation der Zwischensprache CIMIC/P fuer den PEARL-BASIS-SUBSET [1] soll ein erster Schritt in diese Richtung getan werden. Durch den z.Z. bei der GPP mbH in Entwicklung befindlichen PEARL-Compiler werden PEARL-Programme in CIMIC/P-Programme umgesetzt, bevor die dann mit relativ einfachen Codegeneratoren in den Assembler oder die Binder/Laderform eines Zielrechners ueberfuehrt werden.

Um den ohnedies fuer jeden Zielrechner zu erstellenden Codegenerator auch bei der Bereitstellung des PEARL-Compilers einsetzen zu koennen, wird der PEARL-Compiler ebenfalls in der Zwischensprache dargestellt. Dabei wird eine kleine Untermenge CIMIC/C von CIMIC/P verwendet, um auch die indirekte Programmierung in PEARL, (z.B. fuer Mikrorechensysteme) die einen Codegenerator fuer den Gastrechner erforderlich macht, kostenguenstig zu ermoeglichen.

Die vorliegende Spezifikation beschreibt die Zwischensprache CIMIC/C, in welcher der PEARL-Compiler fuer den Arbeitskreis "PEARL fuer Kleinrechner" (PFK) dargestellt wird.

2. Die abstrakte Maschine

CIMIC/C kann als Assemblersprache einer abstrakten Maschine angesehen werden. Zur Vereinfachung der Darstellung von CIMIC/C wird nachfolgend die Struktur und die Arbeitsweise dieser Maschine insoweit beschrieben, als sie mittels CIMIC/C-Anweisungen angesprochen und beeinflusst werden kann.

Es sei vorab bemerkt, dass die Eigenschaften des hier umrissenen Automaten gewaehlt wurden, um die Algorithmen, die bei der Implementierung hoeherer Programmiersprachen eingesetzt werden, auf einem relativ niedrigen Abstraktionslevel [2] darstellen zu koennen. Es ist keinesfalls erforderlich, dass eine reale Rechenanlage, die ueber CIMIC/C erreicht werden soll, mit einem Assoziativspeicher oder einem Operandenstack der hier verwendeten Art ausgeruestet sein muss. Die Adressvergabe und die Abbildung des Stacks kann bei der Codegenerierung erledigt werden und obliegt dem Implementator des Codegenerators.

2.1 Aufbau der abstrakten Maschine

Die dem Compiler-Set zugrundegelegte abstrakte Maschine ergibt sich aus der von W.M. Waite vorgeschlagenen und fuer CIMIC eingesezten JANUS-Maschine [3,4,5] durch starke Vereinfachung des Prozessors. Von den fuer die Adressierung vorgesehenen Registern wird fuer CIMIC/C nur das BASIS-Register eingesetzt. Die Anzahl der Operationen und verarbeitbaren Operandentypen wurde aus in Abschnitt 1. genannten Gruenden stark abgemagert.

Die CIMIC/C-Maschine besteht aus einem Prozessor, einem Speicher und einem Operanden-Stack (vergl. Abb. 1). Der Speicher und der Stack bestehen jeweils aus adressierbaren Einheiten, die u.a. zur Aufnahme von Ganzzahlen und Adressen geeignet sind.

Im Speicher der abstrakten Maschine wird Platz fuer Befehle und benannte Operanden vorgesehen. Sofern diese innerhalb von Rechenoperationen auftreten, werden sie zuvor in den Operanden-Stack gebracht, wo auch die unbenannten Operanden, z.B. die waehrend der Auswertung von arithmetischen Ausdruecken auftretenden Zwischenergebnisse, Platz finden. Der Operanden-Stack wird streng im LAST-IN/FIRST-OUT-Mode betrieben. Je nach der Art des Operators (monadisch oder dyadisch) wird nur auf die oberste bzw. die beiden obersten Stackzellen zugegriffen.

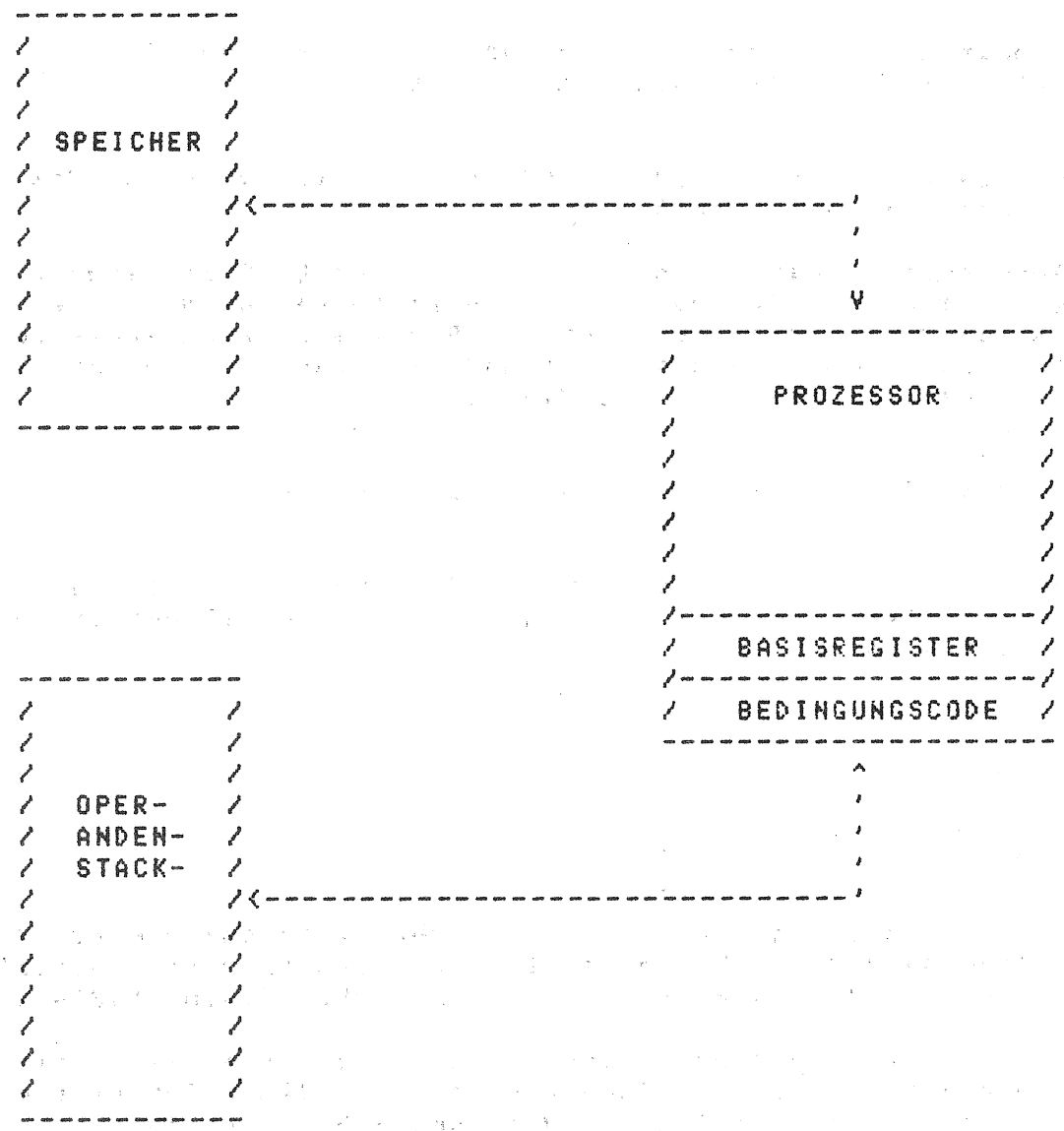

Abb.1: Struktur der CIMIC/C-Maschine

2.2 Ausfuehrung von Rechenoperationen

Eine Rechenoperation besteht aus einem Operator und, falls der rechte Operand noch nicht im Stack vorliegt, aus einem expliziten Operanden.

Sie wird in folgenden vier Schritten ausgefuehrt:

a) Falls ein expliziter Operand angegeben ist, wird er in den Stack geschafft. Andernfalls entfaellt Schritt a).

b) Eine vom Operator abhaengige Anzahl von Objekten wird aus dem Stack in den Prozessor ueberfuehrt.

   Bei monadischen Operatoren wird das oberste Stackelement in den Prozessor gebracht. Bei dyadischen Operatoren werden die beiden obersten Stackelemente ueberfuehrt. Dabei wird vorausgesetzt, dass sich der linke bzw. rechte Operand in der zweitobersten bzw. obersten besetzten Stackzelle befindet.

c) Der Prozessor fuehrt die angegebene Operation aus.

d) Das Ergebnis der Operation wird aus dem Prozessor in den Stack kopiert. Dieser Schritt kann durch Operatormodifikation (N-Flag) unterdrueckt werden. ((1))

2.3 Registerverwendung

Neben den Arbeitsregistern, auf die hier nicht eingegangen wird, da sie in CIMIC/C-Anweisungen nicht explizite ansprechbar sind, enthaelt der Prozessor das sog. BASIS-Register und ein Register BEDINGUNGSCODE.

Das BASIS-Register wird zur Adressierung des Speichers der abstrakten Maschine eingesetzt. Die Ausfuehrung einer TRANSFER-Operation vom Speicher in den Operanden-Stack oder umgekehrt erfolgt in zwei Schritten:

a) Falls ein expliziter Operand angegeben ist, wird seine Adresse in das BASIS-Register gebracht. Andernfalls enthaelt das BASIS-Register schon die Operandenadresse und Schritt a) entfaellt. ((2))

b) Die Transfer-Operation wird durchgefuehrt.

Bei der Durchfuehrung von Vergleichsanweisungen wird das Vergleichsergebnis im Register BEDINGUNGSCODE notiert. Abhaengig vom Inhalt des Registers kann dann der Programmablauf in nachfolgenden bedingten Sprunganweisungen verzweigt werden.

Folgende Vergleichsergebnisse sind in BEDINGUNGSCODE abspeicherbar:

LT Der Bedingungscode LT wird erzeugt, wenn der linke Operand kleiner als der rechte Operand ist.

EQ Der Bedingungscode EQ wird bei Gleichheit der Operanden erzeugt.

GT Der Bedingungscode GT wird erzeugt, wenn der linke Operand groesser als der rechte Operand ist.

3. Konstituenten von CIMIC/C-Anweisungen

3.1 Zum Aufbau von CIMIC/C-Anweisungen

Zur Darstellung der Operationen, die von der in Abschnitt 2 beschriebenen abstrakten Maschine ausgefuehrt werden, benoetigt man in der Regel drei Angaben:

a) den Operationscode, der bestimmt, welche Operation bzw. Pseudooperation ausgefuehrt werden soll,

b) den Operandentyp, der angibt, in welcher Weise der Operand bei der Ausfuehrung der Operation zu interpretieren ist und

c) die Operandenangabe, die das Objekt einfuehrt bzw. den Ort, an dem es zu finden ist, angibt.

Bei bestimmten Operationscodes fehlt die Angabe b) und/oder die Angabe c). Beispielsweise kann die Operandenangabe entfallen, wenn sich der Operand schon im Stack befindet (vergl. Abschnitt 3.4).

3.2 Operationscodes

Die CIMIC/C-Anweisungen koennen in Anlehnung an die uebliche Einteilung von Assembler-Instruktionen in Pseudo-Anweisungen und in eigentliche Anweisungen eingeteilt werden. Im folgenden werden, der Uebersicht halber, die in CIMIC/C vorgesehenen Operationscodes aufgelistet. Ihre genaue Wirkungsweise wird bei der Beschreibung der Anweisungen und Pseudoanweisungen angegeben.

Pseudo-Codes: ARYND

BEGIN

BLEND

BLOCK

CONT

ELMV

END

ENDBLK

GLOBAL

LOC

MODEND

MODULE

PARAM

RPAREND

SPACE

SPCFY

TAEND

TASK

Transfer-Codes: BASE

LOAD

STORE

TEST

Rechen-Codes: ADD

BAND

BNOT

BOR

CMP

DIV

REM

MPY

NEG

SHIFTL

SHIFTR

SUB

Steuer-Codes: ABORT

ARGIS

BEGCAS

CASE

ENDCAS

JMP

RCALL

RCEND

RRETURN

STOP

Einige Operationscodes koennen durch Anhaengen von Flags modifiziert werden. Ein "KEIN-RESULTAT" - Flag:

(N)

bedeutet, dass Schritt d) der in Abschnitt 2.2 beschriebenen Auswertungsvorschrift entfaellt, d.h. es werden nach Durchfuehrung der Operation durch den Prozessor keine Werte in den CIMIC/C-Stack transferiert.

Beispiel: STORE INT BASED:

In diesen Fall enthaelt das Basis-Register die Adresse der Speicherzelle, in welche der in der obersten Stackzelle befindliche Wert ueberfuehrt werden soll. Der Wert wird

in den Stack zurueckgespeichert.

STORE(N) INT BASED;

Wie im vorhergehenden Beispiel; der Wert wird nicht in den Stack zurueckgespeichert.

3.3 Operandentypen

Der Typ eines Operanden bestimmt die Interpretation des Operanden bei der Verarbeitung in der abstrakten Maschine. Bei der Uebersetzung von CIMIC/C vermittelt der Operandentyp dem Codegenerator den Zugriff zu den Eigenschaften eines Operanden (z.B. Speicherbedarf zur Darstellung der Operanden). Die Eigenschaften werden bei der Erstellung des Codegenerators festgelegt.

In CIMIC/C sind folgende Operandentypen vorgesehen:

MODE: 'STR' / S-MODE.

S-MODE: 'ADDR' / 'INT' .

Sie bedeuten:

ADDR — Ein Operand vom Typ ADDR stellt eine Speicheradresse im Daten- oder Anweisungsbereich eines Programms dar.

Um Adressarithmetik in einfacher Weise zu ermoeglichen, wird in CIMIC/C vorausgesetzt, dass diese unter Verwendung der Ganzzahl-Arithmetik abgewickelt werden kann.

STR — Ein Operand vom Typ STR ist die interne Darstellung einer Zeichenkette.

INT — Ein Operand vom Typ INT ist die interne Darstellung einer positiven oder negativen ganzen Zahl.

Fuer die Abbildung von CIMIC/C auf eine reale Maschine wird vorausgesetzt, dass Ganzzahlen Z im Bereich:

-32768 kleiner gleich Z kleiner gleich 32767

darstellbar sind.

3.4 Operanden

In CIMIC/C wird zwischen elementaren Operanden und zusammengesetzten Operanden unterschieden. Elementar heisst ein Operand, wenn ueber seine Darstellung in der Zwischensprache keine Aussage gemacht werden kann. Elementar sind solche Operanden, die normalerweise unmittelbar auf Operanden realer Rechenanlagen abbildbar sind.

Zusammengesetzte Operanden werden in CIMIC aus elementaren Operanden aufgebaut. In CIMIC/C sind Felder elementarer Operanden vorgesehen. Ein Feld ist eine Folge von n > 0 Elementen, die alle vom gleichen Datentyp sind. Elemente eines Feldes werden in ausfuehrbaren Anweisungen ueber Indizes I im Bereich

0 kleiner gleich I kleiner gleich N - 1

angesprochen.

Ein Operand besitzt die allgemeine Form:

OPERAND:

DATA-CONSTANT /

['I,'] VARIABLE.

Eine Konstante DATA-CONSTANT stellt einen Wert dar, der vor der Ausfuehrung eines Programmsystems bekannt ist und waehrend der Ausfuehrung eines Programmsystems nicht veraendert wird. Konstante koennen bei der Erstellung eines CIMIC/C- Programmsystems, bei der Erstellung des Codegenerators fuer eine Zielmaschine oder beim Binden/Laden eines Programmsystems festgelegt werden.

Um die Festlegung von Konstanten vor und waehrend der Phasen eines Uebersetzungsvorgangs zu ermoeglichen, sind in CIMIC/C mehrere Formen von Konstantenangaben vorgesehen:

DATA-CONSTANT:

DENOTATION /

MANIFEST .

Mittels DENOTATIONen werden Konstanten dargestellt, die bei der Erstellung eines CIMIC/C- Programmsystems festgelegt werden. Die unter Verwendung von DENOTATIONen angegebenen Konstanten stellen sich selbst dar. Keine weitere Information ausser der DENOTATION wird benoetigt, um den Operanden zu erkennen:

DENOTATION:

SIMPLE-DENOTATION /

STRING-DENOTATION.

Ganzzahl- und Adresskonstanten werden unter SIMPLE-DENOTATION zusammengefasst.

SIMPLE-DENOTATION:

INTEGER-CONSTANT /

'R ' LABEL.

Folgende Denotationen sind in CIMIC/C vorgesehen:

INTEGER-CONSTANT:

'AINT ' ['-'] DIGIT*

stellt eine Konstante vom Typ INT im Bereich -32768 bis 32767 dar.

'R ' LABEL

stellt eine (abstrakte) Adresse im Befehlskoerper einer Task oder

Prozedur dar.

```
STRING-DENOTATION:
'S ' INTEGER [ (',' INTEGER) *
                STRING-CONTINUATION $ ].

STRING-CONTINUATION:
     '/' EOL
     'CONT STR S ' (INTEGER ',') $ INTEGER.
```

stellt eine Zeichenketten-Konstante dar. Die Zeichen werden als Ganzzahlen im ASCII-Code dargestellt.

Um ein in CIMIC/C vorliegendes Programmsystem auf eine Zielmaschine zu ueberfuehren, ist nicht nur die Abbildung der CIMIC-Anweisungen auf Anweisungen der Zielmaschine auszufuehren, sondern es sind auch bestimmte maschinenabhaengige Konstanten festzulegen. Maschinenabhaengige Konstanten werden im Programm ueber

```
MANIFEST: 'M ' IDENTIFIER.
```

angesprochen. Darin steht IDENTIFIER fuer den Bezeichner einer Konstanten, deren Wert bei der Codegenerierung festgelegt wird. In CIMIC/C werden folgende Manifestwerte verwendet:

M TRUE

M FALSE — Eine der Konstanten geht aus der anderen durch boolsche Negierung hervor.

M ADRSHIFT — Um den PEARL-Compiler auch einfach an Rechenanlagen mit Byte-Adressierung adaptieren zu koennen, wird angenommen, dass sich die Adressen zweier aufeinander folgender Speicherworte zur Aufnahme von Ganzzahlen oder Adressen um eine Potenz von 2 mit Exponent M ADRSHIFT voneinander unterscheiden ((8)).

Speicherplaetze fuer Operanden, die waehrend des Ablaufs eines Programmsystems unterschiedliche Werte annehmen koennen (VARIABLE), muessen mittels CIMIC/C-Anweisungen explizite angefordert werden (vergl. Abschnitt 4.2). Fuer VARIABLEn wird Speicherplatz im Speicher der abstrakten Maschine vorgesehen. Um den verschiedenartigen Speicher-Vergabe-Strategien und Speicher-Zugriffs-Mechanismen Rechnung zu tragen, sind mehrere Speicher-Kategorien vorgesehen:

```
VARIABLE:

      ('GLOBAL '  XSYMB /

      ('STATIC' / 'DISP0')  ' ' SYMBOL /

      'BASED'.
```

Eine Variable besteht danach aus einer Kategorie und gegebenenfalls aus einem Speicherplatz-Bezeichner SYMBOL bzw. XSYMB. Waehrend SYMBOL bzw. XSYMB den in der Speicherzelle gespeicherten Wert anspricht, wird durch die Kategorie der Zugriff zu dieser Speicherzelle festgelegt.

Die angegebenen Speicherkategorien bedeuten im einzelnen:

GLOBAL	Speicherplatz fuer globale Variablen bleibt waehrend des gesamten Ablaufs eines Programmsystems erhalten und ist in allen Moduln, aus denen ein Programmsystem erzeugt wird, ansprechbar.
STATIC	Speicherplatz fuer statische Variablen ist nur in dem Modul, in dem er vereinbart wird, ansprechbar.
DISP0	SYMBOL wird als Speicherplatz einer lokalen Variablen im Aktivierungsbereich einer Task interpretiert. Der dynamisch anzulegende Bereich fuer lokale Prozedurvariablen gehoert zum Aktivierungsbereich der Task, unter deren Regie die Prozedur ablaeuft. Innerhalb einer Prozedur sind in CIMIC/C nur globale Variable, statische Variable, lokale Prozedurvariable und Parameter ansprechbar, d.h. der dynamische Aufbau eines Displayvektors eruebrigt sich ((3)).
BASED	Das BASIS-Register enthaelt die Adresse einer Variablen.

Durch Voranstellen des I-Flags, kann statt des Wertes einer Variablen deren Adresse angesprochen werden.

Beispiel: LOAD ADDR I,LOCAL $L127;

Die Adresse der lokalen Variablen $L127 wird in den Operanden-Stack geladen.

Neben den eigentlichen Variablen sind fuer die kurzzeitige Speicherung von Zwischenergebnissen anonyme temporaere Variablen vorgesehen.

Anonyme temporaere Variable sind Operanden, die nach ihrer Erzeugung genau einmal verwendet werden. Sie treten normalerweise waehrend der Auswertung von Ausdruecken auf. Jeder Operator verarbeitet eine vorgegebene Anzahl von Operanden und erzeugt eine bestimmte Anzahl von Ergebnissen (siehe Abschnitt 2.2). Fuer die Verarbeitung anonymer temporaerer Variablen dient der Operanden-Stack. Er wird angesprochen, wenn die Operandenangabe fehlt (Default).

4. Pseudo-Anweisungen

4.1 Moduln

Entsprechend der fuer PEARL vorgesehenen Modultechnik [6,7,8] kann ein CIMIC/C-Programmsystem aus einer Anzahl unabhaengig voneinander erstellter Moduln aufgebaut werden. Jeder Modul beginnt mit der CIMIC/C- Pseudoanweisung

'MODULE ' EOL

und endet mit der Pseusoanweisung:

'MODEND' EOL

Jeder Modul besteht aus einer Folge von Vereinbarungen:

MODULE:

'MODULE ' EOL

([MODULE-ELEMENT] EOL)$

'MODEND' EOL.

MODULE-ELEMENT:

GLOBAL-DECLARATION-OR-SPECIFICATION /

STATIC-DECLARATION /

PROCEDURE-DECLARATION /

TASK-DECLARATION.

Die Deklaration bzw. Spezifikation globaler Groessen erfolgt in CIMIC nach der Syntax:

```
GLOBAL-DECLARATION-OR-SPECIFICATION:

     'GLOBAL ' ( PROCEDURE-OR-TASK-ENTRY /

                 ' ' ( GLOBAL-DECLARATION /

                       GLOBAL-SPECIFICATION ) ) .
```

Fuer globale Prozeduren und Tasks werden in dem Modul, in dem sie vereinbart sind, modul-lokale abstrakte Adressen angegeben:

```
PROCEDURE-OR-TASK-ENTRY:

     XSYMB  ' '  ENTRY-POINT.

ENTRY-POINT:

     ('PRENTR ' / 'TAENTR ')  SYMBOL.
```

Durch XSYMB wird der globale Bezeichner einer Prozedur oder Task definiert, ueber den sie anderen Moduln angesprochen werden kann. SYMBOL bezeichnet den lokalen Namen (abstrakte Adresse) der Prozedur oder Task in dem Modul, in dem sie vereinbart ist.

Beispiel: Die globale Task PEARL besitzt die lokale abstrakte Adresse $T1:

```
MODULE ;

     .
     .
     .

GLOBAL PEARL TAENTR $T1;

     .
     .
     .

TASK  GLOBAL $T1;

     .
     .
     .

TAEND;
```

.
.
.

MODEND;

Die Deklaration von globalen Variablen erfolgt gemaess:

```
GLOBAL-DECLARATION:
     'DSECT '  EOL
          ('SPACE INT '  GLOBAL-DECLARATOR EOL)*
          'ENDBLK  DSECT '.
```

Zwischen der einleitenden GLOBAL-Anweisung und der den Datenblock abschliessenden ENDBLK-Anweisung werden die Elemente des Datenblocks vereinbart. In CIMIC/C gibt es nur globale Variable vom Typ INT, fuer die mittels der SPACE-Anweisung Speicherplatz reserviert wird. Auf diesen Speicherplatz kann in anderen Moduln ueber den Bezeichner XSYMB fuer die globale Groesse zugegriffen werden:

```
GLOBAL-DECLARATOR:  'DSECT '  XSYMB.
```

Um auf eine globale Groesse in einem Modul, in dem sie nicht vereinbart ist, zugreifen zu koennen, muss sie in diesem Modul durch eine Spezifikation eingefuehrt werden:

```
GLOBAL-SPECIFICATION:
     'REFER '  EOL
          ( 'SPCFY ' ( 'INT ' GLOBAL-DECLARATOR /
                       'ADDR X ' XSYMB ) EOL ) *
     'ENDBLK  REFER '.
```

Beispiel: Die Spezifikationszeile der globalen Variablen GL besitzt die Form:

```
SPCFY INT DSECT GL.
```

Die globale Prozedur P wird mit der Zeile

```
SPCFY ADDR X P.
```

spezifiziert.

4.2 Block- und Variablenvereinbarung

Alle CIMIC/C-Variablen sind zu vereinbaren. Die Vereinbarung fuehrt den Bezeichner SYMBOL fuer die Variable ein und spezifiziert alle relevanten Eigenschaften der Variablen.

Statische Variablen werden gemaess:

```
STATIC-DECLARATION:
     'SPACE ' MODE ' STATIC ' SYMBOL
          ( ' '  DATA-CONSTANT /
            ELEM-COUNT ( EOL /
                         '+' EOL ARRAY-VALUE )).
```

vereinbart.

Bei einfachen statischen Variablen muss ein Anfangswert nach dem Bezeichner fuer die einfache Variable SYMBOL angegeben werden.

Bei Feldern gibt ELEM-COUNT die Anzahl der Feldelemente an:

```
ELEM-COUNT:  '(' INTEGER ')'.
```

Die Initialisierung von Feldelementen wird durch ein auf ELEM-COUNT folgendes '+' angezeigt.

Darauf folgen die Anfangswerte fuer die Feldelemente:

```
ARRAY-VALUE:
     ('ELMV '  MODE  ' STATIC (1) '
          DATA-CONSTANT EOL)$
     'ARVND '  MODE  ' STATIC (1) '
          DATA-CONSTANT.
```

Jede ELMV-Anweisung definiert den Anfangswert fuer jeweils ein Feldelement. Die Feldvereinbarung wird durch eine ARVND-Anweisung abgeschlossen, in der dem Feldelement mit dem groessten Index ein Anfangswert zugewiesen wird.

Beispiel: Vereinbarung eines statischen Feldes von Ganzzahlen mit 3 Elementen. Die Anfangswerte fuer die Feldelemente lauten der Reihe nach: 3,4,5.

```
SPACE INT STATIC $L29(3)+.
ELMV INT STATIC (1) AINT 3.
ELMV INT STATIC (1) AINT 4.
ARVND INT STATIC (1) AINT 5.
```

Die Vereinbarung lokaler Prozedur- und Taskvariablen erfolgt nach der Produktionsregel:

```
LOCAL-DECLARATION:
     'SPACE '  S-MODE  ' DISP0 ' SYMBOL [ ELEM-COUNT ] /
     LOCAL-BLOCK-DECLARATION .
```

Darin dient die erste Alternative zur Deklaration von einfachen lokalen Variablen und von lokalen Feldern. Durch die Gesamtheit der lokalen Variablendeklarationen innerhalb einer Prozedur wird der Speicherbedarf fuer die lokalen Prozedurdaten festgelegt. Die Vergabe und Freigabe dieses Speicherplatzes erfolgt innerhalb des Aktivierungsbereiches der Task, unter deren Regie die Prozedur ausgefuehrt wird, dynamisch zur Laufzeit bei Eintreten bzw. Verlassen einer Prozedur.

Die zweite Alternative:

```
LOCAL-BLOCK-DECLARATION:

     'BLOCK '  INTEGER  ' DISP0 ' EOL

          (CODE-ELEMENT EOL)*

     'BLEND '  INTEGER  ' DISP0 '.
```

ermoeglicht u.a. eine oekonomische Verwendung des Speicherplatzes fuer lokale Prozedurdaten (Blockstruktur).

Durch die Block-Pseudoanweisung wird eine Anfangsadresse innerhalb des Aktivierungsbereichs definiert, relativ zu dem die innerhalb des Blocks vereinbarten lokalen Variablen angesprochen werden. Die BLEND-Anweisung gibt den fuer die lokalen Variablen zugeordneten Speicherplatz wieder frei. Um die Schachtelung lokaler Bloecke zu ermoeglichen, wird durch die in der BLOCK- bzw. BLEND-Anweisung angegebene Ganzzahl-Konstante INTEGER die Blocktiefe bzgl. der Prozedur oder Task festgelegt, die den Block enthaelt. Der aeusserste lokale Block besitzt die Blocktiefe 1. Prozeduren und Tasks liegen auf Blockniveau 0 ((4)).

Beispiel: Die folgende Prozedur $127 benoetigt 2 lokale Speicherplaetze zur Aufnahme von Ganzzahlen.

```
.
.
.

BEGIN  DISP0 $L27;

BLOCK 1 DISP0 ;

SPACE INT DISP0 $L50;

.
.
```

```
                .
        BLOCK 2 DISP0 ;
        SPACE INT DISP0 $L57;
                .
                .
                .
        BLEND 2 DISP0 ;
        BLOCK 2 DISP0 ;
        SPACE INT DISP0 $L80;
                .
                .
                .
        BLEND 2 DISP0 ;
                .
                .
                .
        BLEND 1 DISP0 ;
        END  PROC $L27;
                .
                .
                .
```

4.3 Task- und Prozedurvereinbarung

Ein CIMIC/C-Modul besteht (wie ein PEARL-Modul) aus einer Anzahl von Spezifikationen und Vereinbarungen. Die in Abschnitt 5 bis 7 beschriebenen ausfuehrbaren Anweisungen sind Bestandteile der Task- und Prozedurvereinbarungen.

Eine Taskvereinbarung besitzt die Form:

TASK-DECLARATION:

'TASK ' TASK-HEADING EOL

(CODE-ELEMENT EOL)*

'TAEND'.

In der ersten Zeile der Taskvereinbarung wird der lokale Name der Task angegeben:

TASK-HEADING:

' DISPØ ' SYMBOL.

Darauf folgt das Segment der Task:

CODE-ELEMENT:

LOCAL-DECLARATION /

LABEL-DECLARATION /

STATEMENT.

Eine Taskdeklaration wird mit einer TAEND- Anweisung abgeschlossen ((5)).

Prozedurvereinbarungen bestehen aus einem Prozedurkopf und einem Prozedurkoerper:

PROCEDURE-DECLARATION:

PROCEDURE-HEAD PROCEDURE-BODY.

Der Prozedurkopf enthaelt alle Informationen, die fuer das Anbinden der Prozedur an eine Aufrufumgebung benoetigt werden:

PROCEDURE-HEAD:

'BEGIN ' PROCEDURE-HEADING EOL

('PARAM INT DISP0 ' SYMBOL EOL)$

'RPAREND ' PROCEDURE-HEADING EOL.

Aus der ersten Zeile des Prozedurkopfs:

PROCEDURE-HEADING:

[INT] ' DISP0 ' SYMBOL.

geht hervor,

. ob die Prozedur einen Wert an die aufrufende Prozedur oder Task zurueckliefert (Funktionsprozeduren) und

. unter welchem lokalen Namen SYMBOL die Prozedur innerhalb des Moduls angesprochen wird.

Bei Funktionsprozeduren wird der Funktionswert in der obersten besetzten Zelle des Operandenstack uebergeben.

Falls an eine Prozedur beim Aufruf Parameter uebergeben werden, folgen auf die BEGIN- Anweisung die Spezifikationen der formalen Parameter. Sie definieren jeweils den Parametertyp und die lokalen Namen, unter denen die Parameter in der Prozedur angesprochen werden. Alle Parameter werden per Wert uebertragen.

Der Prozedurkopf wird mit der RPAREND-Anweisung abgeschlossen, mit der die Prozedur in die aktuelle Programmumgebung eingebunden wird.

Der Prozedurkoerper besteht aus Deklarationen und ausfuehrbaren Anweisungen:

PROCEDURE-BODY:

```
(CODE-ELEMENT EOL)*

'END '  PROCEDURE-HEADING.
```

Der Speicherbedarf fuer lokale Variable ist in CIMIC/C zur Uebersetzungszeit bekannt. Speicherplatz fuer lokale Variable wird zusammen mit dem Speicherplatz fuer aktuelle Parameter im Aktivierungsbereich der Task vorgesehen, unter der die Prozedur ausgefuehrt wird.

4.4 Markenvereinbarung

Eine Marke wird im CIMIC-Programmtext mit der Anweisung:

```
LABEL-DECLARATION:

        'LOC '  SYMBOL.
```

eingefuehrt. Darin stellt SYMBOL die abstrakte Adresse der naechsten CIMIC-Anweisung dar. Bei der Ausfuehrung einer LOC-Anweisung ist der Operanden-Stack leer.

5. Transfer-Anweisungen

Transfer-Anweisungen bewegen Information innerhalb der abstrakten Maschine. In CIMIC/C sind in Transfer-Anweisungen nur elementare Operanden zugelassen.

'LOAD ' S-MODE ' ' OPERAND

Der durch OPERAND spezifizierte Wert wird in den Prozessor und von dort in den Operanden-Stack kopiert.

'STORE' ['(N)'] ' ' S-MODE ' ' VARIABLE

Das oberste Element des Operanden-Stack wird in den Prozessor und von dort in das durch VARIABLE angegebene Speicherelement uebertragen. Wenn kein N-Flag angegeben wurde, wird das Element danach in den Operandenstack zuruecktransferiert.

'TEST ' CONDITION-CODE ' '

Wenn der im Register BEDINGUNGSCODE gespeicherte Wert mit dem in der TEST-Anweisung angegebenen CONDITION-CODE uebereinstimmt, wird durch den Prozessor der Wert M TRUE erzeugt. Andernfalls wird der Wert M FALSE generiert. Der erzeugte Wert wird in den Operanden-Stack ueberfuehrt.

'BASE ADDR '

Der Inhalt der obersten Stackzelle wird in das BASIS-Register gebracht und nicht in den Stack zurueckkopiert.

6. Algorithmische Anweisungen

Eine Rechenanweisung besitzt die allgemeine Form:

```
COMPUTATION:
    OPERATION ' ' S-MODE ' ' [ OPERAND ].
```

Folgende OPERATIONen sind in CIMIC/C vorgesehen:

'ADD'	Die Summe der Operanden wird im Operandenstack abgelegt.
'BAND'	Die Operanden werden bitweise mit logischem Und verknuepft und das Ergebnis im Stack gespeichert.
'BNOT'	Der Operand wird bitweise invertiert und das Ergebnis im Stack gespeichert.
'BOR'	Die Operanden werden bitweise mit logischem Oder verknuepft und das Ergebnis im Stack abgelegt.
'CMP' ['(N)']	Die Operanden werden verglichen. Im Register BEDINGUNGSCODE wird der Wert LT bzw. EQ bzw. GT gespeichert, wenn der linke Operand kleiner bzw. gleich bzw. groesser als der rechte Operand ist. Falls das N-Flag nicht angegeben ist, wird der linke Operand in den Stack zurueckgespeichert.
'DIV'	Der linke Operand wird durch den rechten Operanden dividiert und der gegebenenfalls gegen Null gerundete ganzzahlige Teil des Quotienten wird im Stack abgelegt.
'SHIFTL'	Der linke Operand wird um so viele Bitpositionen nach links geschoben, wie der rechte Operand angibt. Von rechts werden

Nullen nachgezogen. Das Ergebnis wird im Stack gespeichert.

'REM' Der linke Operand wird durch den rechten Operanden dividiert und der Rest im Stack abgelegt.

'MPY' Das Produkt der Operanden wird im Stack abgelegt.

'NEG' Der Operand wird negiert und in den Stack zurueckgespeichert.

'SHIFTR' Der linke Operand wird um die Anzahl von Bitpositionen nach rechts geschoben, die der rechte Operand angibt. Von links werden Nullen nachgezogen. Das Ergebnis wird im Stack gespeichert.

'SUB' Der rechte Operand wird vom linken Operanden subtrahiert und das Resultat in den Stack gebracht.

7. Anweisungen fuer die Ablaufsteuerung

7.1 Sprunganweisungen

Mittels Sprunganweisungen kann die sequentielle Ausfuehrung von CIMIC/C-Anweisungen unbedingt oder bedingt unterbrochen und der Ablauf einer Task an einer vorgegebenen Stelle fortgesetzt werden.

In einer Sprunganweisung wird ein Sprungziel und wahlweise ein CONDITION-CODE angegeben:

```
'JMP '  [ CONDITION-CODE ]
     ' '  ( 'R '  LABEL / VARIABLE )
```

Die folgende Tabelle gibt zu jedem der sechs zugelassenen CONDITION-CODES die Werte im Register BEDINGUNGSCODE an, die zur Durchfuehrung der bedingten Sprunganweisung fuehren:

CONDITION-CODE	Wert im Register BEDINGUNGSCODE
'EQ'	EQ
'NE'	LT oder GT
'LT'	LT
'LE'	LT oder EQ
'GE'	EQ oder GT
'GT'	GT

Wird eine bedingte Sprunganweisung ausgefuehrt, so ist danach der Wert im Register BEDINGUNGSCODE undefiniert. Andernfalls bleibt der Wert in BEDINGUNGSCODE erhalten.

Der Taskablauf wird unbedingt verzweigt, wenn kein CONDITION-CODE angegeben wurde.

7.2 Fallauswahl

In einer Fallauswahl wird eine Anzahl von Adressen:

'R ' LABEL

angegeben, von denen eine aufgrund eines Vergleichs ausgewaehlt wird. Jeder Adresse ist eine vorgegebener Ganzzahl-Konstante zugeordnet. Wenn der Inhalt des obersten besetzten Stackelements mit einer der Konstanten uebereinstimmt, wird der Taskablauf bei der zugeordneten Adresse fortgesetzt. Andernfalls wird zu der Adresse verzweigt, die in der die Fallauswahl abschliessenden Anweisung angegeben ist.

Eine Fallauswahl wird durch die Anweisung:

'BEGCAS INT ' EOL

eingeleitet. Durch sie wird das oberste Stackelement in den CIMIC/C-Prozessor transferiert und nicht in den Stack zurueckkopiert. Auf die BEGCAS-Anweisung folgen Anweisungen, in denen Adressen mit Ganzzahl-Konstanten verknuepft werden:

'CASE INT R ' LABEL ' ' INTEGER-CONSTANT EOL

Die in einer Fallauswahl angegebenen Konstanten muessen saemtliche voneinander verschieden sein.

Die Fallauswahl wird mit der Anweisung:

'ENDCAS ADDR R ' LABEL

beendet. Wie oben schon vermerkt, wird zu der darin angegebenen Marke verzweigt, wenn der Wert des in den Prozessor kopierten obersten Stackelements mit keiner der Ganzzahl-Konstanten uebereinstimmt.

7.3 Beenden von Tasks

Eine Task kann entweder normal mit der STOP-Anweisung oder abnormal mit der ABORT-Anweisung beendet werden.

Beispiele:

```
STOP;

ABORT;
```

7.4 Prozeduraufruf

Ein Prozeduraufruf ermoeglicht eine Programmverzweigung mit automatischer Rueckkehr an die Aufrufstelle. An die Prozedur koennen Parameter uebergeben werden und die Prozedur kann einen Wert an die aufrufende Umgebung zurueckliefern. Eine Aufrufsequenz enthaelt alle Anweisungen zur Verzweigung des Taskablaufs und fuer die Berechnung der aktuellen Parameter.

Sie beginnt mit der Anweisung:

```
'RCALL '  [ 'INT' ]  ' '  TARGET EOL

TARGET:

     'X ' XSYMB /

     VARIABLE.
```

durch die ein rekursiver Prozeduraufruf der durch TARGET bezeichneten Prozedur eingeleitet wird.

Die Typangabe INT ist bei Funktionsprozeduren anzugeben.

Auf die RCALL-Anweisung folgen gegebenenfalls Anweisungen zur Berechnung aktueller Parameter. Dabei sind alle Transfer-Anweisungen, algorithmische Anweisungen und Anweisungen fuer die Ablaufsteuerung zugelassen. Insbesondere kann eine Aufrufsequenz weitere Aufrufsequenzen enthalten.

Jeder aktuelle Parameter wird durch:

'ARGIS INT ' EOL

spezifiziert.

Mit der Anweisung:

'RCEND ' ['INT'] ' ' TARGET

wird eine Aufrufsequenz beendet. Wie bei der RCALL-Anweisung ist die Typangabe nur bei Funktionsprozeduren erforderlich.

7.5 Verlassen einer Prozedur

Die Rueckkehr aus einer Prozedur in die aufrufende Prozedur bzw. Task wird durch die Anweisung:

'RRETURN ' PROCEDURE-HEADING

bewirkt.

Durch die RETURN-Anweisung wird der Zeiger im Aktivierungsbereich der Task, unter deren Regie die Prozedur ausgefuehrt wird, auf den Stand

zurueckgesetzt, den er vor dem Prozeduraufruf besass.

8. CIMIC/C-Programmbeispiel

Das folgende Programmbeispiel demonstriert das Aussehen eines CIMIC/C-Programmoduls. Es kann bei Verfuegbarkeit eines Codegenerators fuer CIMIC/C und nach Realisierung der aeusserst einfach gewaehlten Ausgabe-Schnittstelle ausgefuehrt werden und liefert dann den weiter unten aufgefuehrten Ausgabetext auf einem Druckgeraet ((6)).

CIMIC/C-Programmbeispiel

```
MODULE ;
SPACE ADDR STATIC $L103 R $L102;
SPACE ADDR STATIC $L105 R $L104;
SPACE ADDR STATIC $L107 R $L106;
SPACE ADDR STATIC $L109 R $L108;
SPACE ADDR STATIC $L111 R $L110;
SPACE ADDR STATIC $L113 R $L112;
SPACE STR STATIC $L118 S 9,65,85,83,71,65,66,69,32,68,69,83,32,67/;
CONT STR S 73,77,73,67,47,67,45,80,82,79,71,82,65,77/;
CONT STR S 77,66,69,73,83,80,73,69,76,83,13,32,9,45/;
CONT STR S 45,45,45,45,45,45,45,45,45,45,13,45,45,45/;
CONT STR S 45,45,45,45,45,45,45,45,45,45,45,45,45,45/;
CONT STR S 45,45,45,45,45,45,45,45,45;
SPACE INT STATIC $L119(6);
SPACE ADDR STATIC $L120(6)+;
ELMV ADDR STATIC (1) R $L123;
ELMV ADDR STATIC (1) R $L124;
ELMV ADDR STATIC (1) R $L125;
ELMV ADDR STATIC (1) R $L126;
ELMV ADDR STATIC (1) R $L127;
ARVND ADDR STATIC (1) R $L128;
SPACE STR STATIC $L128 S 69,73,78,32,84,69,73,76,32,68,69,82,32,65/;
CONT STR S 82,73,84,72,77,69,84,73,75,46;
SPACE STR STATIC $L127 S 80,82,79,71,82,65,77,77,86,69,82,90,87,69/;
CONT STR S 73,71,85,78,71,69,78,32,85,78,68;
SPACE STR STATIC $L126 S 68,73,69,32,70,65,76,76,65,85,83,87,65,72/;
CONT STR S 76,44;
SPACE STR STATIC $L125 S 69,73,78,73,71,69,32,83,67,72,76,69,73,70/;
CONT STR S 69,78,84,89,80,69,78,32,40,75,79,80,70,45/;
CONT STR S 32,85,78,68,32,83,67,72,87,65,78,90,65,66/;
CONT STR S 70,82,65,71,69,41,44;
```

```
SPACE STR STATIC $L124 S 68,73,69,32,80,65,82,65,77,69,84,69,82,85/;
CONT STR S 69,66,69,82,71,65,66,69,44;
SPACE STR STATIC $L123 S 71,69,84,69,83,84,69,84,32,87,73,82,68,32/;
CONT STR S 68,73,69,32,65,85,83,71,65,66,69,32,65,85/;
CONT STR S 70,32,69,73,78,69,77,32,68,82,85,67,75,69/;
CONT STR S 82,44;
SPACE STR STATIC $L129 S 68,82,85,67,75,69,82;
SPACE STR STATIC $L130 S 13,49;
SPACE STR STATIC $L135 S 46,32,32;
SPACE STR STATIC $L136 S 13,32;
SPACE STR STATIC $L137 S 13,32,9,90,65,69,72,76,83,67,72,76,69,73/;
CONT STR S 70,69,32,85,78,68,32,90,85,71,82,73,70,70/;
CONT STR S 32,65,85,70,32,70,69,76,68,69,76,69,77,69/;
CONT STR S 78,84,32,71,69,84,69,83,84,69,84;
SPACE STR STATIC $L138 S 9,84,69,83,84,32,68,69,82,32,80,65,82,65/;
CONT STR S 77,69,84,69,82,85,69,66,69,82,71,65,66,69/;
CONT STR S 32,32,80,82,79,90,69,68,85,82;
SPACE STR STATIC $L139 S 9,84,69,83,84,32,68,69,82,32,80,65,82,65/;
CONT STR S 77,69,84,69,82,85,69,66,69,82,71,65,66,69/;
CONT STR S 32,32,70,85,78,75,84,73,79,78,83,80,82,79/;
CONT STR S 90,69,68,85,82;
SPACE STR STATIC $L141 S 9,80,65,82,65,77,69,84,69,82,85,69,66,69/;
CONT STR S 82,71,65,66,69,32,79,46,32,75,46;
SPACE STR STATIC $L142 S 9,83,67,72,76,69,73,70,69,78,84,69,83,84/;
CONT STR S 83;
SPACE STR STATIC $L147 S 13,32,13,32,9,80,82,79,71,82,65,77,77,83/;
CONT STR S 67,72,76,69,73,70,69,78,32,79,46,32,75,46;
SPACE STR STATIC $L148 S 9,84,69,83,84,32,70,65,76,76,65,85,83,87/;
CONT STR S 65,72,76;
SPACE ADDR STATIC $L151 R $L150;
SPACE STR STATIC $L156 S 9,70,65,76,76,32,69,82,75,65,78,78,84;
SPACE STR STATIC $L158 S 9,70,65,76,76,65,85,83,87,65,72,76,32,79/;
CONT STR S 46,32,75,46;
SPACE STR STATIC $L159 S 9,84,69,83,84,32,86,69,82,90,87,69,73,71/;
CONT STR S 85,78,71,69,78,32,85,78,68,32,65,85,83,68/;
CONT STR S 82,85,69,67,75,69,44,76,69,84,90,84,69,82/;
CONT STR S 69,83,32,84,69,73,76,87,69,73,83,69;
SPACE INT STATIC $L161(4);
ELMV INT STATIC (1) AINT 1;
ELMV INT STATIC (1) AINT 2;
ELMV INT STATIC (1) AINT 3;
ARVND INT STATIC (1) AINT 4;
SPACE STR STATIC $L165 S 9,65,85,83,68,82,85,67,75,83,65,85,83,87/;
CONT STR S 69,82,84,85,78,71,32,79,68,69,82,32,13,32/;
CONT STR S 9,86,69,82,90,87,69,73,71,85,78,71,32,70/;
CONT STR S 69,72,76,69,82,72,13,65,70,84;
SPACE STR STATIC $L166 S 9,65,85,83,68,82,85,67,75,83,65,85,83,87/;
CONT STR S 69,82,84,85,78,71,32,85,78,68,32,86,69,82/;
CONT STR S 90,87,69,73,71,85,78,71,32,79,46,32,75,46;
SPACE STR STATIC $L167 S 69,78,68,69,32,68,69,82,32,76,73,83,84,69;
SPACE STR STATIC $L181 S 13,32;
SPACE INT STATIC $L193(5);
ELMV INT STATIC (1) AINT 1;
```

```
ELMV INT STATIC (1) AINT 10;
ELMV INT STATIC (1) AINT 100;
ELMV INT STATIC (1) AINT 1000;
ARVND INT STATIC (1) AINT 10000;
GLOBAL  DSECT ;
SPACE INT DSECT G101;
ENDBLK  DSECT ;
GLOBAL  REFER ;
SPCFY ADDR X G1;
SPCFY ADDR X G2;
SPCFY ADDR X G3;
ENDBLK  REFER ;
GLOBAL BEISPIEL TAENTR $L114;
BEGIN  DISP0 $L102;
RPAREND  DISP0 $L102;
BLOCK 1 DISP0 ;
SPACE INT DISP0 $L116;
SPACE INT DISP0 $L117;
LOAD ADDR I,STATIC $L118;
SHIFTR INT M ADRSHIFT;
STORE(N) INT DISP0 $L116;
BLOCK 2 DISP0 ;
SPACE INT DISP0 $L121;
LOAD INT AINT 0;
STORE(N) INT DISP0 $L121;
LOC $L122;
LOAD INT DISP0 $L121;
SHIFTL INT M ADRSHIFT;
ADD ADDR I,STATIC $L120;
BASE ADDR ;
LOAD INT BASED;
SHIFTR INT M ADRSHIFT;
LOAD INT DISP0 $L121;
SHIFTL INT M ADRSHIFT;
ADD ADDR I,STATIC $L119;
BASE ADDR ;
STORE(N) INT BASED;
LOAD INT DISP0 $L121;
ADD INT AINT 1;
STORE INT DISP0 $L121;
CMP(N) INT AINT 6;
JMP LT R $L122;
BLEND 2 DISP0 ;
LOAD ADDR I,STATIC $L119;
SHIFTR INT M ADRSHIFT;
STORE(N) INT DISP0 $L117;
RCALL INT X G1;
LOAD ADDR I,STATIC $L129;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND INT X G1;
STORE(N) INT GLOBAL G101;
RCALL  STATIC $L111;
```

```
LOAD ADDR I,STATIC $L130;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L111;
LOAD INT DISP0 $L116;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 4;
ARGIS INT ;
RCEND  STATIC $L109;
RCALL  STATIC $L109;
LOAD INT AINT 4;
ARGIS INT ;
RCEND  STATIC $L109;
BLOCK 2 DISP0 ;
SPACE INT DISP0 $L133;
SPACE INT DISP0 $L134;
LOAD INT AINT 0;
STORE(N) INT DISP0 $L133;
JMP  R $L131;
LOC $L132;
RCALL  STATIC $L113;
LOAD INT AINT 1;
ADD INT DISP0 $L134;
ARGIS INT ;
RCEND  STATIC $L113;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L135;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L111;
LOAD INT DISP0 $L134;
ADD INT DISP0 $L117;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L136;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
LOAD INT DISP0 $L134;
ADD INT AINT 1;
STORE(N) INT DISP0 $L133;
LOC $L131;
LOAD INT DISP0 $L133;
STORE INT DISP0 $L134;
CMP(N) INT AINT 5;
```

```
JMP LE R $L132;
BLEND 2 DISP0 ;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L137;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 3;
ARGIS INT ;
RCEND  STATIC $L109;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L138;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 2;
ARGIS INT ;
RCEND  STATIC $L109;
RCALL  STATIC $L105;
LOAD INT DISP0 $L117;
ARGIS INT ;
RCEND  STATIC $L105;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L139;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 2;
ARGIS INT ;
RCEND  STATIC $L109;
BLOCK 2 DISP0 ;
SPACE INT DISP0 $L140;
RCALL INT STATIC $L107;
LOAD ADDR I,DISP0 $L117;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND INT STATIC $L107;
STORE(N) INT DISP0 $L140;
RCALL  STATIC $L105;
LOAD INT DISP0 $L140;
ARGIS INT ;
RCEND  STATIC $L105;
BLEND 2 DISP0 ;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L141;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 3;
```

```
ARGIS INT ;
RCEND  STATIC $L109;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L142;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 2;
ARGIS INT ;
RCEND  STATIC $L109;
BLOCK 2 DISP0 ;
SPACE INT DISP0 $L143;
LOAD INT AINT -1;
STORE(N) INT DISP0 $L143;
LOC $L144;
LOAD INT AINT 1;
ADD INT DISP0 $L143;
STORE(N) INT DISP0 $L143;
RCALL  STATIC $L111;
LOAD INT DISP0 $L143;
ADD INT DISP0 $L117;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 1;
ARGIS INT ;
RCEND  STATIC $L109;
LOAD INT DISP0 $L143;
CMP(N) INT AINT 5;
JMP LT R $L144;
RCALL  STATIC $L109;
LOAD INT AINT 2;
ARGIS INT ;
RCEND  STATIC $L109;
JMP  R $L146;
LOC $L145;
RCALL  STATIC $L111;
LOAD INT DISP0 $L143;
ADD INT DISP0 $L117;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 1;
ARGIS INT ;
RCEND  STATIC $L109;
LOAD INT DISP0 $L143;
```

```
SUB INT AINT 1;
STORE(N) INT DISP0 $L143;
LOC $L146;
LOAD INT DISP0 $L143;
CMP(N) INT AINT -1;
JMP NE R $L145;
BLEND 2 DISP0 ;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L147;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 3;
ARGIS INT ;
RCEND  STATIC $L109;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L148;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 2;
ARGIS INT ;
RCEND  STATIC $L109;
BLOCK 2 DISP0 ;
SPACE INT DISP0 $L149;
LOAD INT AINT 0;
STORE(N) INT DISP0 $L149;
LOC $L150;
JMP  R $L152;
LOC $L155;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L156;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 2;
ARGIS INT ;
RCEND  STATIC $L109;
LOAD INT AINT 1;
STORE(N) INT DISP0 $L149;
JMP  STATIC $L151;
LOC $L157;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L158;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 3;
ARGIS INT ;
```

```
RCEND  STATIC $L109;
JMP  R $L154;
LOC $L152;
LOAD INT DISP0 $L149;
BEGCAS INT ;
CASE INT R $L155 AINT 0;
ENDCAS ADDR R $L157;
LOC $L154;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L159;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 2;
ARGIS INT ;
RCEND  STATIC $L109;
BLOCK 3 DISP0 ;
SPACE INT DISP0 $L160;
LOAD ADDR I,STATIC $L161;
SHIFTR INT M ADRSHIFT;
STORE(N) INT DISP0 $L160;
BLOCK 4 DISP0 ;
SPACE INT DISP0 $L162;
LOAD ADDR I,DISP0 $L160;
SHIFTR INT M ADRSHIFT;
STORE(N) INT DISP0 $L162;
LOAD INT DISP0 $L160;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
MPY INT AINT 3;
LOAD INT AINT 2;
ADD INT DISP0 $L160;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
SUB INT ;
LOAD INT AINT 3;
ADD INT DISP0 $L160;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
LOAD INT AINT 1;
ADD INT DISP0 $L160;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
DIV INT ;
ADD INT ;
CMP(N) INT AINT 2;
JMP EQ R $L163;
RCALL  STATIC $L111;
```

```
LOAD ADDR I,STATIC $L165;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
JMP  R $L164;
LOC $L163;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L166;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
LOC $L164;
BLEND 4 DISP0 ;
BLEND 3 DISP0 ;
RCALL  STATIC $L109;
LOAD INT AINT 3;
ARGIS INT ;
RCEND  STATIC $L109;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L167;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  X G2;
LOAD INT GLOBAL G101;
ARGIS INT ;
RCEND  X G2;
BLEND 2 DISP0 ;
BLEND 1 DISP0 ;
RRETURN  DISP0 $L102;
END  DISP0 $L102;
BEGIN  DISP0 $L104;
PARAM INT DISP0 $L168;
RPAREND  DISP0 $L104;
BLOCK 1 DISP0 ;
SPACE INT DISP0 $L171;
SPACE INT DISP0 $L172;
LOAD INT AINT 0;
STORE(N) INT DISP0 $L171;
JMP  R $L169;
LOC $L170;
RCALL  STATIC $L111;
LOAD INT DISP0 $L172;
ADD INT DISP0 $L168;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
ARGIS INT ;
RCEND  STATIC $L111;
RCALL  STATIC $L109;
LOAD INT AINT 1;
ARGIS INT ;
RCEND  STATIC $L109;
```

```
LOAD INT DISP0 $L172;
ADD INT AINT 1;
STORE(N) INT DISP0 $L171;
LOC $L169;
LOAD INT DISP0 $L171;
STORE INT DISP0 $L172;
CMP(N) INT AINT 5;
JMP LE R $L170;
BLEND 1 DISP0 ;
RCALL  STATIC $L109;
LOAD INT AINT 3;
ARGIS INT ;
RCEND  STATIC $L109;
RRETURN  DISP0 $L104;
END  DISP0 $L104;
BEGIN INT DISP0 $L106;
PARAM INT DISP0 $L173;
RPAREND INT DISP0 $L106;
LOAD INT DISP0 $L173;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
RRETURN INT DISP0 $L106;
END INT DISP0 $L106;
BEGIN  DISP0 $L108;
PARAM INT DISP0 $L175;
RPAREND  DISP0 $L108;
BLOCK 1 DISP0 ;
SPACE INT DISP0 $L178;
SPACE INT DISP0 $L180;
SPACE INT DISP0 $L179;
LOAD INT DISP0 $L175;
STORE(N) INT DISP0 $L180;
LOAD INT AINT 1;
STORE(N) INT DISP0 $L178;
JMP  R $L176;
LOC $L177;
RCALL  STATIC $L111;
LOAD ADDR I,STATIC $L181;
SHIFTR INT M ADRSHIFT;
ARGIS INT ;
RCEND  STATIC $L111;
LOAD INT DISP0 $L179;
ADD INT AINT 1;
STORE(N) INT DISP0 $L178;
LOC $L176;
LOAD INT DISP0 $L178;
STORE INT DISP0 $L179;
CMP(N) INT DISP0 $L180;
JMP LE R $L177;
BLEND 1 DISP0 ;
RRETURN  DISP0 $L108;
END  DISP0 $L108;
```

```
BEGIN  DISP0 $L110;
PARAM INT DISP0 $L182;
RPAREND  DISP0 $L110;
BLOCK 1 DISP0 ;
SPACE INT DISP0 $L183;
SPACE INT DISP0 $L184;
LOAD INT AINT 8;
STORE(N) INT DISP0 $L183;
LOAD INT AINT 0;
STORE(N) INT DISP0 $L184;
LOC $L185;
LOAD INT DISP0 $L182;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
SHIFTR INT DISP0 $L183;
BAND INT AINT 255;
STORE(N) INT DISP0 $L184;
LOAD INT DISP0 $L184;
CMP(N) INT AINT 0;
JMP NE R $L186;
RRETURN  DISP0 $L110;
LOC $L186;
RCALL  X G3;
LOAD INT GLOBAL G101;
ARGIS INT ;
LOAD INT DISP0 $L184;
ARGIS INT ;
RCEND  X G3;
LOAD INT DISP0 $L183;
SUB INT AINT 8;
STORE(N) INT DISP0 $L183;
LOAD INT DISP0 $L183;
CMP(N) INT AINT 0;
JMP GE R $L187;
LOAD INT AINT 1;
ADD INT DISP0 $L182;
STORE(N) INT DISP0 $L182;
LOAD INT AINT 8;
STORE(N) INT DISP0 $L183;
LOC $L187;
JMP  R $L185;
BLEND 1 DISP0 ;
RRETURN  DISP0 $L110;
END  DISP0 $L110;
BEGIN  DISP0 $L112;
PARAM INT DISP0 $L188;
RPAREND  DISP0 $L112;
LOAD INT DISP0 $L188;
CMP(N) INT AINT 0;
JMP GE R $L189;
RCALL  X G3;
LOAD INT GLOBAL G101;
```

```
ARGIS INT ;
LOAD INT AINT 45;
ARGIS INT ;
RCEND  X G3;
LOAD INT DISP0 $L188;
NEG INT ;
STORE(N) INT DISP0 $L188;
LOC $L189;
LOAD INT DISP0 $L188;
CMP(N) INT AINT 0;
JMP NE R $L190;
RCALL  X G3;
LOAD INT GLOBAL G101;
ARGIS INT ;
LOAD INT AINT 48;
ARGIS INT ;
RCEND  X G3;
RRETURN  DISP0 $L112;
LOC $L190;
BLOCK 1 DISP0 ;
SPACE INT DISP0 $L191;
SPACE INT DISP0 $L192;
LOAD ADDR I,STATIC $L193;
SHIFTR INT M ADRSHIFT;
STORE(N) INT DISP0 $L191;
LOAD INT AINT 1;
STORE(N) INT DISP0 $L192;
JMP  R $L195;
LOC $L194;
LOAD INT AINT 1;
ADD INT DISP0 $L192;
STORE(N) INT DISP0 $L192;
LOC $L195;
LOAD INT DISP0 $L192;
ADD INT DISP0 $L191;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
CMP(N) INT DISP0 $L188;
JMP GT R $L196;
LOAD INT DISP0 $L192;
CMP(N) INT AINT 4;
JMP LE R $L194;
LOC $L196;
LOC $L197;
LOAD INT DISP0 $L192;
SUB INT AINT 1;
STORE(N) INT DISP0 $L192;
BLOCK 2 DISP0 ;
SPACE INT DISP0 $L198;
SPACE INT DISP0 $L199;
LOAD INT AINT -1;
STORE(N) INT DISP0 $L198;
```

```
LOAD INT DISP0 $L192;
ADD INT DISP0 $L191;
SHIFTL INT M ADRSHIFT;
BASE ADDR ;
LOAD INT BASED;
STORE(N) INT DISP0 $L199;
LOC $L200;
LOAD INT DISP0 $L188;
SUB INT DISP0 $L199;
STORE(N) INT DISP0 $L188;
LOAD INT AINT 1;
ADD INT DISP0 $L198;
STORE(N) INT DISP0 $L198;
LOAD INT DISP0 $L188;
CMP(N) INT AINT 0;
JMP GE R $L200;
RCALL  X G3;
LOAD INT GLOBAL G101;
ARGIS INT ;
LOAD INT AINT 48;
ADD INT DISP0 $L198;
ARGIS INT ;
RCEND  X G3;
LOAD INT DISP0 $L199;
ADD INT DISP0 $L188;
STORE(N) INT DISP0 $L188;
BLEND 2 DISP0 ;
LOAD INT DISP0 $L192;
CMP(N) INT AINT 0;
JMP NE R $L197;
BLEND 1 DISP0 ;
RRETURN  DISP0 $L112;
END  DISP0 $L112;
TASK  DISP0 $L114;
RCALL  STATIC $L103;
RCEND  STATIC $L103;
STOP;
TAEND;
MODEND;
```

Fuer die Ausfuehrung des Beispiels sind folgende globale Laufzeitfunktionen bereitszustellen, die normalerweise schon in dieser oder einer aehnlichen Form verfuegbar sind ((7)):

Prozedurname	Bedeutung
G1	Die Funktionsprozedur G1 oeffnet das Ausgabefile. Als Parameter wird die um M ADRSHIFT nach rechts geschobene Adresse der Zeichenkette DRUCKER (Title-Option) uebergeben. Der Funktionswert ist maschinenabhaengig und kann z.B. die Adresse des File-Kontroll-Blocks fuer das Standard-Ausgabefile oder die (logische) Kanalnummer eines Druckgeraets sein.
G2	Die Prozedur G2 schliesst das durch G1 geoeffnete Ausgabefile ab und leert dabei ggf. den Ausgabepuffer. Parameter ist der Funktionswert, der beim Oeffnen des Files G1 angeliefert wurde.
G3	schreibt ein Zeichen auf den Ausgabefile. Es werden folgende Parameter uebergeben: 1. das Resultat von G1 (siehe oben) und 2. das auszugebende Zeichen als Ganzzahl im ASCII-Code.

Bei der Ausgabe wird folgende Formatsteuerung eingesetzt:

. Seitenvorschub:	CR (Bytewert 13) gefolgt von 1 (Bytewert 49)
. Zeilenvorschub:	CR (Bytewert 13) gefolgt von Leerzeichen (Bytewert 32)
. Tabulator:	HT (Bytewert 9) positioniert auf die naechste durch 10 teilbare Spaltennummer.

Neben den genannten maschinenabhaengig zu erstellenden Prozeduren enthaelt das Programmbeispiel eine Prozedur $L110 fuer die Ausgabe einer Zeichenkette auf das Standard-Ausgabefile. Diese wurde im Hinblick auf

eine einfache Adaptierung des Programmbeispiels (und des PEARL-Compilers) an einen Prozessrechner mit einer Wortlaenge von 16 Bit erstellt. Falls eine andere Wortlaenge vorliegt oder falls eine der nachfolgend angegebenen Voraussetzungen nicht erfuellt ist, muss die Funktion bei der Adaption des Programmbeispiels modifiziert oder ersetzt werden.

Die Ausgabe-Funktion $L110 fuer Zeichenketten arbeitet unter folgenden Voraussetzungen:

- ein Zeichen wird durch 8 Bit dargestellt.
- Zeichenketten werden gepackt. Jeweils 2 Zeichen werden in einem Wort abgelegt. Das zuerst auszugebende Zeichen wird im linken Byte eines Wortes abgelegt.
- Das Ende der Zeichenkette wird durch den Wert NUL angegeben (Bytewert 0).

Die Ausfuehrung des oben aufgelisteten CIMIC/C-Programmbeispiels fuehrt zu folgender Ausgabeliste:

```
          AUSGABE DES CIMIC/C-PROGRAMMBEISPIELS
          -------------------------------------
 1.  GETESTET WIRD DIE AUSGABE AUF EINEM DRUCKER,
 2.  DIE PARAMETERUEBERGABE,
 3.  EINIGE SCHLEIFENTYPEN (KOPF- UND SCHWANZABFRAGE),
 4.  DIE FALLAUSWAHL,
 5.  PROGRAMMVERZWEIGUNGEN UND
 6.  EIN TEIL DER ARITHMETIK.
          ZAEHLSCHLEIFE UND ZUGRIFF AUF FELDELEMENT GETESTET
          TEST DER PARAMETERUEBERGABE  PROZEDUR
 GETESTET WIRD DIE AUSGABE AUF EINEM DRUCKER,
 DIE PARAMETERUEBERGABE,
 EINIGE SCHLEIFENTYPEN (KOPF- UND SCHWANZABFRAGE),
 DIE FALLAUSWAHL,
 PROGRAMMVERZWEIGUNGEN UND
 EIN TEIL DER ARITHMETIK.
          TEST DER PARAMETERUEBERGABE  FUNKTIONSPROZEDUR
 GETESTET WIRD DIE AUSGABE AUF EINEM DRUCKER,
 DIE PARAMETERUEBERGABE,
 EINIGE SCHLEIFENTYPEN (KOPF- UND SCHWANZABFRAGE),
 DIE FALLAUSWAHL,
 PROGRAMMVERZWEIGUNGEN UND
 EIN TEIL DER ARITHMETIK.
          PARAMETERUEBERGABE O. K.
          SCHLEIFENTESTS
 GETESTET WIRD DIE AUSGABE AUF EINEM DRUCKER,
```

```
DIE PARAMETERUEBERGABE,
EINIGE SCHLEIFENTYPEN (KOPF- UND SCHWANZABFRAGE),
DIE FALLAUSWAHL,
PROGRAMMVERZWEIGUNGEN UND
EIN TEIL DER ARITHMETIK.
EIN TEIL DER ARITHMETIK.
PROGRAMMVERZWEIGUNGEN UND
DIE FALLAUSWAHL,
EINIGE SCHLEIFENTYPEN (KOPF- UND SCHWANZABFRAGE),
DIE PARAMETERUEBERGABE,
GETESTET WIRD DIE AUSGABE AUF EINEM DRUCKER,
          PROGRAMMSCHLEIFEN O. K.
          TEST FALLAUSWAHL
          FALL ERKANNT
          FALLAUSWAHL O. K.
          TEST VERZWEIGUNGEN UND AUSDRUECKE,LETZTERES TEILWEISE
          AUSDRUCKSAUSWERTUNG UND VERZWEIGUNG O. K.
ENDE DER LISTE
```

Fussnoten

((1)) Das N-Flag ist nicht bei allen Rechenoperationen sinnvoll. Die Operationen, bei denen Flags erlaubt sind, koennen aus der Syntaxnotation (siehe Anhang B) entnommen werden.

((2)) Mit der BASE-Operation kann eine Adresse aus dem Stack in das BASIS-Register geladen werden.

((3)) Mit anderen Worten: es gibt nur Prozeduren auf Modulebene.

((4)) Die Speichervergabe fuer lokale Prozedurdaten erfolgt bei der Codegenerierung relativ zum aktuellen Belegungszeiger fuer den Aktivierungsbereich. Letzterer wird bei Eintreten bzw. Verlassen einer Prozedur dynamisch gesetzt bzw. zurueckgesetzt.

((5)) Fuer die Steuerung von Tasks sind in CIMIC/C keine Anweisungen vorgesehen, da diese fuer den PEARL-Compiler nicht benoetigt werden. Es wird vorausgesetzt, dass globale Tasks mittels des fuer PEARL zu erstellenden Bediensystems, ueber Steuerkarten oder per Bedienanweisung aktivierbar sind.

((6)) Aus Aufwandsgruenden mussten bei der Erstellung des Beispiels auf einen systematischen Funktionstest, wie er fuer die Verifikation von Codegeneratoren erforderlich ist, verzichtet werden.

((7)) Da die Ein/Ausgabe in CIMIC/C ueber globale maschinenabhaengige Laufzeitfunktionen ausgefuehrt wird, ist

sie nicht Bestandteil der vorliegenden Spezifikation. Die zwecks Adaption des PEARL-Compilers fuer den BASIS-Subset erforderlichen elementaren E/A-Funktionen werden, in aehnlicher Weise wie in dem Beispiel, bei der Auslieferung des PEARL-Compilers in einem Adaptionsblatt beschrieben.

((8)) Die Konstante M ADRSHIFT wird nur in SHIFT-Anweisungen verwendet. Ist der Adressabstand zwischen zwei aufeinander folgenden Speicherworten zur Aufnahme von Ganzzahlen gleich 1 bzw. ist M ADRSHIFT gleich 0 (Wort-Adressierung), so sind alle CIMIC/C-Anweisungen, in denen M ADRSHIFT vorkommt, bei der Codegenerierung zu eliminieren.

Literaturangaben

[1] B.F. Eichenauer:
Spezifikation des PEARL-BASIS-SUBSET
Teil 1: Syntax des PEARL-BASIS-SUBSET
Muenchen, den 8. Juli 1976
GPP mbH

[2] W.M.Waite:
Software Portability via an Intermediate Language
GFK-GI-GMR Fachtagung Prozessrechner 1974
Karlsruhe, 10.-11. Juni 1974
erschienen in:
Lecture Notes in Computer Science, Bd.12,
Springer Verlag 1974

[3] B.Eichenauer, A.Muehlhahn, P.C.Poole,
J.Reh und W.M.Waite:
CIMIC/1 Vorlaeufige Spezifikation
PDV-Entwicklungsnotizen, PDV-E 15 (1973)

[4] M.C.Newey, P.C.Poole and W.M.Waite:
Abstract Machine Modelling to produce

portable Software - a Rewiev and Evaluation.
Software - Practice and Experience, Vol.2,
187-136 (1972)

[5] S.S.Coleman, P.C.Poole and W.M.Waite:
The mobile Programming System, JANUS.
Software - Practice and Experience, Vol.4,
5-23 (1974)

[6] K.Timmesfeld et.al.:
PEARL, A Proposal for a Process- and
Experiment Automation Realtime Language,
Bericht KFK-PDV 1
Gesellschaft fuer Kernforschung mbH Karlsruhe
April 1973

[7] Ueberarbeitung von PEARL zur Erhoehung der
Uebereinstimmung mit PL/1
Bericht PDV-E 13
Gesellschaft fuer Kernforschung mbH Karlsruhe
Januar 1974

[8] B.Eichenauer et.al.:
PEARL, eine prozess- und experimentorientierte
Programmiersprache
Angewandte Informatik 9 73

Anhang A: Kurze Beschreibung der Notation zur Darstellung der Syntax von CIMIC/C.

Die nachfolgende beschriebene Notation zur Darstellung der Syntax von CIMIC/C unterscheidet sich nur geringfuegig von BNF und lehnt sich an eine von DeRemer und Frank eingefuehrte Darstellungsmethode an.

A.1 Endsymbole

Ein Endsymbol des Vokabulars wird entweder durch einen Bezeichner, ein Sonderzeichen oder durch ein Literal dargestellt.

A.1.1 Bezeichner

Ein Bezeichner ist eine Folge von Buchstaben, Ziffern und Bindestrichen. Er beginnt und endet mit einem Buchstaben oder einer Ziffer; unmittelbar aufeinander folgende Bindestriche sind nicht zugelassen.

Bezeichner werden zur Identifizierung von Grammatikregeln (vergl. Abschnitt A.2) oder zur Identifizierung von Lexikalsymbolen, d.h. von Symbolen, deren Aufbau innerhalb der Grammatik nicht beschrieben wird, eingesetzt. Treten in einer Grammatik Lexikalsymbole auf, so sind diese nach dem Wortsymbol LEXIKAL-SYMBOLE aufzulisten. Die bei der Beschreibung von CIMIC/C verwendeten Lexikalsymbole sind in Abschnitt B.1 beschrieben.

A.1.2 Sonderzeichen

Die folgenden Sonderzeichen werden zur Darstellung von Operatoren eingesetzt:

: * / + $ ( ) [ ] .

A.1.3 Literale

Literale sind sich selbst definierende Zeichenfolgen. Ein Literal wird in Hochkommata eingeschlossen. Das Hochkomma selbst wird durch zwei aufeinander folgende Hochkommata dargestellt.

A.1.4 Zwischenraeume

Aufeinander folgende Endsymbole sind durch Zwischenraeume gegeneinander abzugrenzen. Ist die Identifizierung aufeinander folgender Endsymbole auch ohne Einschub von Zwischenraeumen moeglich, so duerfen diese entfallen.

A.2 Produktionsregeln

Zwischen den Wortsymbolen GRAMMATIK-REGELN und FINIS wird eine Folge von Produktionsregeln angegeben, mit denen die Syntax von CIMIC/C beschrieben wird.

A.2.1 Aufbau einer Produktionsregel

Eine Produktionsregel besteht aus:

- einem Bezeichner (Regelnamen), ueber den der zugeordnete Regelkoerper in den Regelkoerpern anderer Produktionsregeln angesprochen werden kann,
- einem Doppelpunkt zur Abgrenzung des Regelnamens vom Koerper der Regel,
- den Regelkoerper, in dem die oder ein Teil der darzustellenden Syntax notiert ist und
- einem Punkt, mit dem die Produktionsregel beendet wird.

Es gelten folgende Einschraenkungen:

- Jeder Regelname identifiziert genau einen Regelkoerper.
- Genau ein Regelname wird in keinem Regelkoerper verwendet (Startregel).

A.2.2 Alternativen

Alternativen innerhalb eines Regelkoerpers werden durch Schraegstriche gegeneinander abgegrenzt.

Beispiel: A: B / C.

A produziert entweder B oder C.

A.2.3 Optionen

Stimmen zwei Alternativen bis auf eine bestimmte Symbolfolge ueberein, so kann die fragliche Symbolfolge durch Einschliessen in eckige Klammern als optional gekennzeichnet werden.

Beispiel: Die Produktionsregel

```
A: B C / B.
```

kann unter Verwendung von eckigen Klammern auch in der Form:

```
A: B [ C ].
```

dargestellt werden.

A.2.4 Folgen

Eine Folge eines Symbols, in der das Symbol mindestens einmal auftreten muss, kann durch Angabe des Symbols mit nachfolgendem Sternzeichen dargestellt werden:

Beispiel: Die Produktionsregel:

```
A: B [A ].
```

kann auch in der Form:

```
A: B* .
```

geschrieben werden.

Ist auch die Nullfolge zugelassen, so wird statt des Sternzeichens ein Dollarzeichen verwendet.

Beispiel:

```
A: [ B* ].
```

ist gleichbedeutend mit:

A: B$.

A.2.5 Unterteilung

Haeufig treten in Produktionsregeln Aneinandereihungen bestimmter Endsymbole (Symbolketten) auf, die durch eine aus anderen Endymbolen bestehende Kette gegeneinander abgegrenzt sind. Syntaxmuster dieser Art lassen sich bequem unter Verwendung des Operators // darstellen, der als "unterteilt durch" zu lesen ist.

Beispiel: Die Regel

A: B / B ',' A.

produziert: B

B,B

B,B,B

.

.

.

und kann unter Verwendung des

Operators // in der Form:

A: B // ','.

geschrieben werden.

A.2.6 Wahlweise Kettung

Mit dem Operator + kann festgelegt werden, dass sowohl die vor dem Operator, als auch die nach ihm angegebene Symbolkette alleine auftreten darf. Treten beide Konstruktionen gleichzeitig auf, so muss die in der Regel angegebene Reihenfolge eingehalten werden.

Beispiel: A: B [C] / C.

ist gleichbedeutend mit:

A: B + C.

A.2.7 Klammerung

Jedes Auftreten eines Regelnamens im Koerper einer Regel kann durch den dem Regelnamen zugeordneten Regelkoerper ersetzt werden. Falls letzterer nicht nur aus einem Symbol besteht, ist er bei der Ersetzung in runde Klammern einzuschliessen. Wurde ein Regelname ueberall in dieser Weise ersetzt, so kann die durch den Regelnamen identifizierte Produktionsregel entfernt werden.

Die beiden Operatoren // und + sind linksassoziativ. Deshalb koennen Klammernpaare in Regeln, die diese Operatoren enthalten, haeufig weggelassen werden:

Beispiel: Die folgenden drei Definitionen von

A sind gleichbedeutend:

A: X // D. X: B // C.

A: (B // C) // D.

A: B // C // D.

Anhang B: Vollstaendige Syntax von CIMIC/C

B.1 Lexikalsymbole

Wie in Abschnitt B.2 angegebene vollstaendige Syntax von CIMIC/C zeigt, werden die folgenden 4 Bezeichner nicht durch die Grammatik-Regeln erfasst:

END-OF-LINE: ist ein Steuerzeichen, das im Eingabestrom auftritt und ueblicherweise einem Wagenruecklauf oder einem Kartenende entspricht.

LETTER: steht fuer ein Zeichen des Alphabets.

DIGIT: steht fuer eine Ziffer.

COMMENT: CHARACTER *.

Darin ist CHARACTER eines der folgenden Zeichen:

CHARACTER: LETTER / DIGIT /

'+' / '-' / '*' / '/' /

'(' / ')' / '.' / ',' /

'=' / ' ' / '$' / ';'.

Zwischenraeume sind in CIMIC relevant. Ihre genaue Position wird durch die in Abschnitt B.2 dargestellten Syntaxregeln angegeben.

B.2 Syntaxregeln

Um ein bequemes Nachschlagen zu ermoeglichen, ist im folgenden die komplette Syntax von CIMIC/C angegeben. Das Auffinden einer bestimmten Produktionsregel wird durch die in Abschnitt B.3 angegebene, bzgl. der Regelnamen alphabetisch geordnete Querbezugsliste erleichtert. In ihr wird jedem Regelnamen u.a. die Zeilennummer in der Syntaxbeschreibung zugeordnet, in welcher der Regelkoerper beginnt.

```
1      LEXIKAL-SYMBOLE
2
3              DIGIT
4              LETTER
5              COMMENT
6              END-OF-LINE
7
8
9
10     GRAMMATIK-REGELN
11
12        MODULE:
13           'MODULE '  EOL
14              ( [ MODULE-ELEMENT ]  EOL   )$
15              'MODEND' EOL.
16
17
18
19        IDENTIFIER: LETTER (LETTER / DIGIT) $.
20
21
22
23        EOL:  ';' [ COMMENT ] END-OF-LINE.
24
25
26
27        MODULE-ELEMENT
28           GLOBAL-DECLARATION-OR-SPECIFICATION /
29           STATIC-DECLARATION /
30           PROCEDURE-DECLARATION /
31           TASK-DECLARATION .
32
33
34
35        GLOBAL-DECLARATION-OR-SPECIFICATION:
36           'GLOBAL ' (  PROCEDURE-OR-TASK-ENTRY /
37                        ' ' ( GLOBAL-DECLARATION /
38                              GLOBAL-SPECIFICATION ) ).
39
```

```
40
41
42          PROCEDURE-OR-TASK-ENTRY:
43             XSYMB ' ' ENTRY-POINT.
44
45
46
47          XSYMB:  IDENTIFIER.
48
49
50
51          ENTRY-POINT:  ( 'PRENTR '  /  'TAENTR '  )  SYMBOL.
52
53
54
55          GLOBAL-DECLARATION:
56               'DSECT '  EOL
57                  ( 'SPACE INT '  GLOBAL-DECLARATOR  EOL )*
58                  'ENDBLK  DSECT ' .
59
60
61
62          GLOBAL-DECLARATOR:  'DSECT '  XSYMB .
63
64
65
66          SYMBOL:  '$'  (LETTER/DIGIT)*.
67
68
69
70          GLOBAL-SPECIFICATION:
71               'REFER '  EOL
72                  ( 'SPCFY ' ( 'INT '  GLOBAL-DECLARATOR /
73                               'ADDR X ' XSYMB )  EOL ) *
74                  'ENDBLK  REFER '.
75
76
77
78          STATIC-DECLARATION:
79             'SPACE '  MODE  ' STATIC '  SYMBOL
80                ( ' '  DATA-CONSTANT  /
81                  ELEM-COUNT (EOL /
82                              '+' EOL ARRAY-VALUE )).
83
84
85
86          MODE:  'STR'  /  S-MODE .
87
88
89
90          S-MODE:  'ADDR'  /  'INT' .
91
92
```

```
93
94        ELEM-COUNT:  '(' INTEGER ')'.
95
96
97
98        INTEGER: DIGIT*.
99
100
101
102       DATA-CONSTANT:
103          DENOTATION / MANIFEST.
104
105
106
107       DENOTATION:
108          SIMPLE-DENOTATION /
109          STRING-DENOTATION .
110
111
112
113       SIMPLE-DENOTATION:
114          INTEGER-CONSTANT /
115          'R '  LABEL .
116
117
118
119       STRING-DENOTATION:
120          'S '  INTEGER [ ( ',' INTEGER )*
121                            STRING-CONTINUA $  ].
122
123
124
125       STRING-CONTINUATION:
126          '/'  EOL
127          'CONT STR S '  ( INTEGER ',' )$ INTEGER.
128
129
130
131       LABEL:   SYMBOL.
132
133
134
135       INTEGER-CONSTANT:
136          'AINT '  [ '-' ]  DIGIT*.
137
138
139
140       MANIFEST:  'M ' IDENTIFIER.
141
142
143
144       ARRAY-VALUE:
145          ('ELMV '  MODE
```

```
146             ' STATIC (1) '  DATA-CONSTANT  EOL )$
147          'ARVND '  MODE
148             ' STATIC (1) '  DATA-CONSTANT .
149
150
151
152       PROCEDURE-DECLARATION:
153          PROCEDURE-HEAD  PROCEDURE-BODY.
154
155
156
157       PROCEDURE-HEAD
158          'BEGIN '  PROCEDURE-HEADING  EOL
159          ( 'PARAM INT DISP0 '  SYMBOL  EOL )$
160          'RPAREND '  PROCEDURE-HEADING  EOL.
161
162
163
164       PROCEDURE-HEADING:
165          [ 'INT' ]  ' DISP0 '  SYMBOL.
166
167
168
169       PROCEDURE-BODY
170          (CODE-ELEMENT  EOL)*
171          'END '  PROCEDURE-HEADING.
172
173
174
175       CODE-ELEMENT:
176          LOCAL-DECLARATION /
177          LABEL-DECLARATION /
178          STATEMENT .
179
180
181
182       LOCAL-DECLARATION:
183          'SPACE '  S-MODE  ' DISP0 '  SYMBOL  [ ELEM-COUNT ] /
184          LOCAL-BLOCK-DECLARATION.
185
186
187
188       LOCAL-BLOCK-DECLARATION:
189          'BLOCK ' INTEGER ' DISP0' EOL
190             (CODE-ELEMENT EOL)*
191          'BLEND ' INTEGER ' DISP0' .
192
193
194
195       LABEL-DECLARATION:  'LOC ' SYMBOL.
196
197
198
```

```
199       STATEMENT:
200          TRANSMISSION /
201          COMPUTATION /
202          CONTROL-TRANSFER.
203
204
205
206       TRANSMISSION:
207          'LOAD ' S-MODE  ' '  OPERAND /
208          'STORE' [ '(N)' ]  ' ' S-MODE  ' '  VARIABLE /
209          'TEST '  CONDITION-CODE  ' '  /
210          'BASE ADDR '.
211
212
213
214       OPERAND:
215          DATA-CONSTANT /
216          [ 'I,' ]  VARIABLE .
217
218
219
220       CONDITION-CODE
221          'LT' / 'LE' / 'EQ' / 'NE' / 'GE' / 'GT' .
222
223
224
225       VARIABLE:
226          'GLOBAL '  XSYMB /
227          ( 'STATIC' / 'DISP0' )  ' '  SYMBOL /
228          'BASED'.
229
230
231
232       COMPUTATION:
233          OPERATION  ' '  S-MODE  ' '  [ OPERAND ] .
234
235
236
237       OPERATION:
238          'ADD' /
239          'BAND' /
240          'BNOT' /
241          'BOR' /
242          'CMP'  [ '(N)' ] /
243          'DIV' /
244          'MPY' /
245          'NEG' /
246          'SHIFTL' /
247          'REM' /
248          'SHIFTR' /
249          'SUB' .
250
251
```

```
252
253       CONTROL-TRANSFER:
254          'JMP '   [ CONDITION-CODE ]
255                   ' ' ( 'R ' LABEL / VARIABLE ) /
256          'BEGCAS INT '   EOL
257             ('CASE INT R '   LABEL
258                INTEGER-CONSTANT EOL)*
259              'ENDCAS ADDR R '   LABEL /
260          CALL /
261          'RRETURN ' PROCEDURE-HEADING /
262          'STOP' /
263          'ABORT' .
264
265
266
267       CALL
268          'RCALL '   [ 'INT' ]   ' '   TARGET   EOL
269             ((CODE-ELEMENT EOL)$
270                'ARGIS INT '   EOL ) $
271             'RCEND '   [ 'INT' ]   ' '   TARGET.
272
273
274
275       TARGET:
276          'X '   XSYMB /
277          VARIABLE .
278
279
280
281       TASK-DECLARATION:
282          'TASK '   TASK-HEADING EOL
283             (CODE-ELEMENT EOL)*
284          'TAEND' .
285
286
287
288       TASK-HEADING:   ' DISP0 '   SYMBOL.
289
290
291
292    FINIS
```

B.3 Querbezugsliste der Regelnamen

Die folgende Querbezugsliste besteht aus 3 Feldern. Im ersten Feld sind unter der Ueberschrift "SYMBOL" die Regelnamen aufgelistet, die bei der Beschreibung von CIMIC/C verwendet werden. Das zweite Feld mit der Ueberschrift "DEFINITION" gibt die Zeilennummer in der Syntaxdarstellung an, in welcher der Regelname definiert ist. Im dritten Feld sind unter der Ueberschrift "VERWENDUNG" die Zeilennummern angegeben, in denen der im ersten Feld angegebene Regelname im Koerper einer Syntaxregel verwendet wird.

SYMBOL	DEFINITION	VERWENDUNG
ARRAY-VALUE	144	82
CALL	267	260
CODE-ELEMENT	175	170,190,269,283
COMPUTATION	232	201
CONDITION-CODE	220	209,254
CONTROL-TRANSFER	253	202
DATA-CONSTANT	102	80,146,148,215
DENOTATION	107	103
ELEM-COUNT	94	81,183
ENTRY-POINT	51	43
EOL	23	13,14,15,56,57,71,73,81,82, 126,146,158,159,160,170,189, 190,256,258,268,269,270,282, 283
GLOBAL-DECLARATION	55	37
GLOBAL-DECLARATION-OR-SPECIFICATION	35	28
GLOBAL-DECLARATOR	62	57,72
GLOBAL-SPECIFICATION	70	38
IDENTIFIER	19	47,140
INTEGER	98	94,120,120,127,127,189,191
INTEGER-CONSTANT	135	114,258
LABEL	131	115,255,257,259
LABEL-DECLARATION	195	177
LOCAL-BLOCK-DECLARATION	188	184
LOCAL-DECLARATION	182	176
MANIFEST	140	103
MODE	86	79,145,147
MODULE	12	NICHT VERWENDET
MODULE-ELEMENT	27	14
OPERAND	214	207,233
OPERATION	237	233
PROCEDURE-BODY	169	153
PROCEDURE-DECLARATION	152	30
PROCEDURE-HEAD	157	153
PROCEDURE-HEADING	164	158,160,171,261