KERNFORSCHUNGSZENTRUM

KARLSRUHE

Juli 1973 KFK 1712

Institut für Datenverarbeitung in der Technik

PLIM – Eine virtuelle PL/1-Maschine

M. Rupp, J. Nehmer, G. Fleck, D. Hilse, R. Friehmelt

GESELLSCHAFT
FÜR
KERNFORSCHUNG M.B.H.

KARLSRUHE

Als Manuskript vervielfältigt

Für diesen Bericht behalten wir uns alle Rechte vor

GESELLSCHAFT FÜR KERNFORSCHUNG M.B.H.
KARLSRUHE

KERNFORSCHUNGSZENTRUM KARLSRUHE

KFK 1712

Institut für Datenverarbeitung in der Technik

PLIM - Eine virtuelle PL/1-Maschine

M. Rupp
J. Nehmer
G. Fleck
D. Hilse
R. Friehmelt

Gesellschaft für Kernforschung m.b.H., Karlsruhe

Kurzfassung

PLIM ist ein Instrument zur Simulation und Analyse der dynamischen Eigenschaften von Betriebssoftware.

Der Bericht beschreibt die Organisation und den Gebrauch des Simulationssystems unter TSO.

Abstract

PLIM - A Virtual PL/1-Machine

PLIM is a simulation facility developped for the analysis of the dynamic behaviour of operating system software.

This report describes the organization and use of PLIM under TSO.

Inhalt: Seite

1. Einleitung

Bei der Entwicklung neuer Rechnersysteme wurde in den letzten Jahren ein ständig wachsender Betrag für die Betriebssoftware bei gleichzeitiger Senkung der Hardwarekosten aufgewendet.

Maßgebliche Ursache für den darin erkennbaren Trend zu komplexeren Betriebsorganisationen zukünftiger DV-Systeme bildet die Evolution der Rechnerhardware vom starren Gerät zur weitgehend flexiblen Konfiguration, die an vorgegebene Aufgaben besser adaptierbar ist.

So können schon heute - wie das Beispiel der PDP-11 zeigt - simple Abrechnungsautomaten ohne nennenswerte Peripherie, Mehrprozessorsysteme sowie Ein/Ausgabe-intensive Konfigurationen mit einer Vielzahl von in der Leistung abgestuften Kanälen und Hintergrundspeichern oder Rechnerverbundnetze aus Grundbausteinen zusammengesteckt werden.

Eine übergeordnete, universelle Betriebsorganisation für das gesamte Spektrum skizzierter Systemkonfigurationen scheint zum gegenwärtigen Zeitpunkt unökonomisch, wenn nicht unmöglich. Mehr Erfolg verspricht die Entwicklung konfigurationsangepaßter Betriebssysteme, die sich strukturell wesentlich voneinander unterscheiden können. Als Beispiel sei der Scheduler eines Einprozessorsystems erwähnt, dessen Planungsstrategien - auch bei ähnlichem Aufgabenprofil - von denen eines Mehrprozessorsystems in der Regel erheblich abweichen werden.

Die Entwicklung alternativer Betriebssystemvarianten sowie die Ermittlung ihrer optimalen Konfigurationsbandbreiten stellen den Entwicklungsingenieur vor neue Probleme.

Die dazu erforderlichen Leistungsmessungen von Systementwürfen lassen sich mit vertretbarem Aufwand nur an Simulationsmodellen durchführen. Sie erlauben bei entsprechender Auslegung eine einfache, auch extrapolierende - d.h. über den beabsichtigten Maximalausbau des realen Systems hinausgreifende - Variation der Hardwarekonfiguration. Simulationstechniken gewinnen daher bei zukünftigen Betriebssystementwicklungen immer mehr an Bedeutung.

2. Anforderungen an ein Simulationssystem

Die Tauglichkeit eines Simulationsinstrumentes soll an 4 Kriterien gemessen werden:

- Bequeme Modellbildung in einer für den Betriebssystementwickler gewohnten Sprache,
- Hoher Dokumentationswert der Modellbeschreibung,
- Niedrige Erstellungskosten,
- Hohe Laufzeiteffizienz.

Problematisch erweist sich der für eine befriedigende Modellebene erforderliche hohe Detailierungsgrad des Modells. Er zwingt dazu, Modelle in einer mit Maschinenprogrammen vergleichbaren Auflösung darzustellen und schließt nach gegenwärtigen Kenntnissen die Benutzung analytischer Methoden praktisch aus. Andererseits sollte möglichst eine höhere Programmiersprache benutzt werden, die eine maschinenunabhängige Darstellung von Algorithmen gestattet und außerdem die Vorteile großer Verbreitung und hohen Dokumentationswertes in sich vereinigt. Der damit vollzogene Vergröberungsprozeß der realen Umwelt, der allen Simulationsmodellen eigen ist, schränkt die möglichen quantitativen Untersuchungen in einer noch näher zu spezifizierenden Weise ein.

Die Wahl einer höheren Programmiersprache als Implementierungsbasis von Betriebssystemmodellen führt unmittelbar zur Vorstellung einer virtuellen Maschine, deren Maschinencode aus Statements dieser Sprache besteht. Versieht man Statements mit Zeitfaktoren, die bei der Programmausführung ausgewertet werden, dann hat man die Voraussetzung für ein Meßverfahren.

Um eine möglichst große Identität von Statements der virtuellen Maschine mit üblichen Maschineninstruktionen zu erreichen, sollten auf der rechten Seite jedes Statements immer nur so viele Operatoren stehen, wie eine durchschnittliche reale Maschine in einer Instruktion verarbeiten kann.

Alle Operationen wirken unmittelbar auf Operanden in einem symbolischen Arbeitsspeicher, da höhere Sprachen in der Regel keine Mittel zur Benutzung von Arbeitsregistern bieten. Eine Optimierung von Algorithmen bezüglich der Arbeitsregister entfällt daher.

Für die weiteren Überlegungen unterstellen wir, daß die benutzte Sprache für verschiedene Organisationsklassen Sprachelemente gleicher Mächtigkeit besitzt (z.B. für Listenmanipulation, String-Handling). Mit dieser Voraussetzung ist das zeitliche Verhalten unterschiedlicher Algorithmen auf der Basis der eingeführten Statement-Zeitfaktoren vergleichbar. Die Ermittlung der Ausführungszeit für ein Statement kann z.B. nach folgender Formel geschehen:

$$T = \left(\sum_{i=1}^{n} a_i + \sum_{j=1}^{m} op_j \right) \cdot \tau$$

τ	Grundtaktzeit der virtuellen Maschine
n	Zahl der Speicherzugriffe zum Abholen der Operanden und Abspeichern des Resultats
m	Anzahl der Operationen
a_i	Zahl der Grundtakte für die Durchführung des i-ten Speicherzugriffs
op_j	Zahl der Grundtakte für die Durchführung der j-ten Operation

(alle Statementbewertungen von Modellprogrammen müssen in ganzen Vielfachen der Grundtaktzeit τ angegeben werden, die eine beliebig wählbare Bezugsgröße ist)

Es ist selbstverständlich, daß die gemessenen absoluten Programmlaufzeiten von derart konstruierten Modellprogrammen lediglich zur Bestimmung einer Größenordnung dienen können. Die quantitativen Untersuchungen beschränken sich daher auf Vergleichsmessungen an alternativen Entwürfen für eine bestimmte Betriebssystemkomponente.

Beim Übergang auf reale Systeme werden durch Registeroptimierungen, algorithmusfreundliche Instruktionsklassen der Zielmaschine, Pipeline-Verhalten oder Pufferspeicher (z.B. CACHE der IBM/370) Verschiebungen in den gewonnenen Vergleichsaussagen auftreten. Je klarer das Resultat jedoch für einen bestimmten Entwurf ausfällt, um so größer wird die Wahrscheinlichkeit dafür, daß die Tendenz - auch für Abbildungen auf unterschiedliche Zielmaschinen - erhalten bleibt. Die Angabe von quantitativen Zusammenhängen ist jedoch außerordentlich schwierig, da sich das Gebiet der Codeoptimierung einer theoretischen Behandlung noch weitgehend verschließt.

3. Übersicht über das Gesamtsystem

Als Basis sowohl für die Implementierung des Simulationssystems als auch für die zu simulierenden Betriebssystemprogramme wurde PL/1 aus folgenden Gründen gewählt:

- Es ist für die Formulierung von Betriebssoftware gut geeignet.
- PL/1-Programme sind leicht lesbar und haben deshalb den geforderten hohen Dokumentationswert.
- In den zentralen DV-Anlagen des KFZK stehen PL/1-Compiler standardmäßig seit längerem zur Verfügung und damit ein relativ großer Kreis erfahrener PL/1-Programmierer.
- Der von IBM eingeführte zeitoptimierende PL/1-Compiler erzeugt Objektprogramme hoher Güte, die eine Grundvoraussetzung für angemessene Simulationslaufzeiten darstellen.
- Die zunehmende Benutzung von PL/1 - auch außerhalb des IBM-Kundenkreises - ermöglicht eine wirksame Verbreitung der erzielten Ergebnisse.

Durch die Wahl einer gemeinsamen Sprache sowohl für die Beschreibung der virtuellen Maschine als auch der Betriebssystemmodelle kann die Hardware/Software-Schnittstelle leicht umdefiniert werden. Das Simulationssystem eignet sich daher

insbesondere auch für die Untersuchung neuartiger Hardwarestrukturen.

Abb. 3-1 zeigt den organisatorischen Aufbau des gesamten Simulationssystems. In der oberen Hälfte ist die Kontrollstruktur dargestellt, die in drei Ebenen untergliedert ist.

Wir unterscheiden

- die Eventsteuerung,
- die Ablaufsteuerung der PL/1-Maschine,
- das Modell.

Event- und Maschinensteuerung zusammen bilden die virtuelle PL/1-Maschine (kurz: PLIM). Das Modell besteht aus dem zu simulierenden Programmsystem und ist aus einer Kollektion von PL/1-Prozeduren (Modellprogramme) zusammengesetzt, an die hinsichtlich des Aufbaus noch näher zu spezifizierende Anforderungen gestellt werden.

Im unteren Teil der Abbildung ist der Erstellungsprozeß für Modellprogramme dargestellt.

Der eigentliche Quellcode wird zunächst durch einen speziellen Makroprozessor (Kap. 5.4.) modifiziert und dann durch den standardmäßigen PL/1 Optimizing Compiler übersetzt.

Eine Besonderheit des hier realisierten Verfahrens ist, daß Modellprogramme entgegen der sonst üblichen Technik nicht interpretiert, sondern statementweise von der PL/1-Maschine durchlaufen werden. Der dadurch notwendige Kontrollflußwechsel zwischen PL/1-Maschine und Modellprogrammen, der nach jedem PL/1-Statement stattfindet, erfordert eine entsprechende Aufbereitung des Quellcodes der Modellprogramme, die durch den bereits erwähnten Makroprozessor vorgenommen wird.

Das damit in groben Zügen skizzierte Verfahren hat folgende Vorteile:

- Anstelle eines speziellen Übersetzers, der Objektcode für die virtuelle Maschine erzeugt, wird lediglich ein vergleichsweise primitiver Makroprozessor benötigt.
- Der Aufwand für einen Codeinterpretierer entfällt.

- Durch den Fortfall der Codeinterpretation wird eine wesentliche Verbesserung der Laufzeiteffizienz erzielt.

4. Syntax der PLIM-Maschinensprache

Für die Sprachbeschreibung der PLIM-Maschinensprache wurde die Syntaxnotation des PL/1-Language Reference Manual / 3 / übernommen. Dadurch wird eine einheitliche und übersichtliche Darstellung der Syntax von Modellprogrammen und der allgemeinen Programmiersprache PL/1 erreicht.

Aufgrund der einheitlichen Syntaxnotation werden hier nur Unterschiede, d.h. Ergänzungen und Einschränkungen der PLIM-Maschinensprache gegenüber der Basis-Sprache PL/1 aufgeführt. Es folgt eine Kurzbeschreibung der Syntaxnotation.

- Notationsvariablen werden mit Kleinbuchstaben bzw. mit grossem Anfangsbuchstaben und nachfolgenden Kleinbuchstaben geschrieben, ggf. enthalten sie auch Ziffern und Sonderzeichen.
- Notationskonstanten sind mit Großbuchstaben und ggf. Ziffern und Sonderzeichen geschriebene Zeichen bzw. Zeichenketten.
- Syntaktische Einheiten sind
  a) einfache Notationsvariable oder Notationskonstante
  b) mehrere Notationsvariablen, Notationskonstanten, Syntaxsymbole und Schlüsselworte, eingeschlossen in geschweiften oder eckigen Klammern.
- Geschweifte Klammern {} fassen eine Gruppe von mehreren Elementen zu einer syntaktischen Einheit zusammen.
  a) Vertikale Schreibweise kennzeichnet Alternativen,
     z.B.:
     ```
     {FIXED}
     {FLOAT}
     ```
  b) Senkrechter Strich | kennzeichnet ebenfalls Alternativen,
     z.B.: {FIXED | FLOAT}

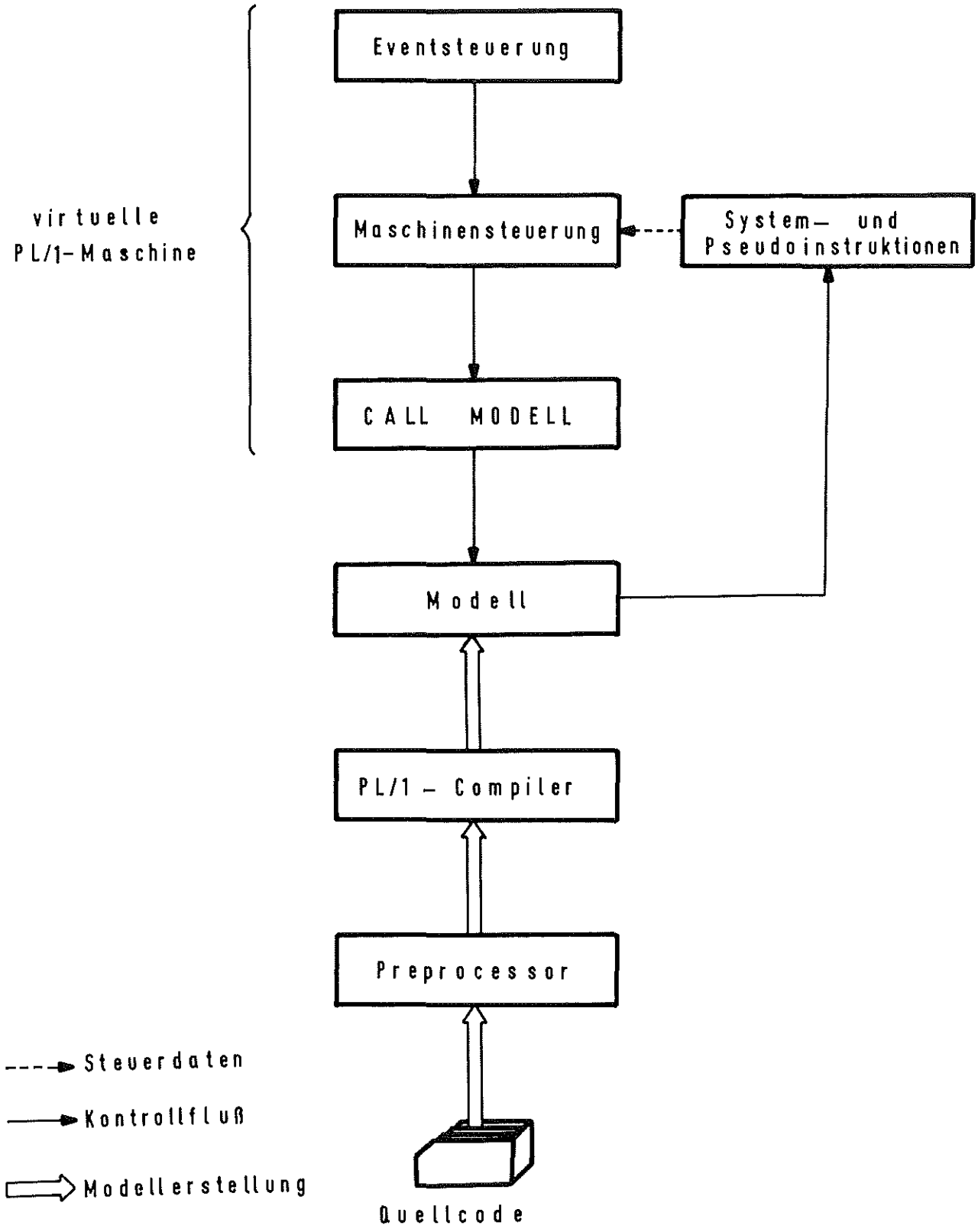


Abb.: 3-1 Organisationsschema des Simulationssystems

- Eckige Klammern [] stellen optionale syntaktische Einheiten dar.
- Drei Punkte ... kennzeichnen einmaliges bzw. mehrfaches Wiederholen der vorausgegangenen syntaktischen Einheit.
- Unterstrichener Text xxx...x kennzeichnet Hinweise oder Sonderstellung von Zeichen bzw. Zeichenketten.

4.1 Syntaktische Beschreibung von Modellprogrammen

Syntax

```
Modellprogramm  →  {Prozedurkopf
                    Deklarations-Statement ...
                    On-Statement   ...
                    PLIM-Instruktion   ...
                    Unterprogramm   ...
                    Prozedurende}
```

Allgemeine Regel:

a) Modellprogramme sind externe Prozeduren, die im Simulationssystem unter der Kontrolle der PLIM ablaufen sollen. Sie werden mit Hilfe des PLIM-Preprocessors (5.4.2.) übersetzt.

b) Die Übergabe von Parametern geschieht mittels externer Namen oder mit Hilfe zweier spezieller Funktionen ($$STOR_P und $$LOAD_P), die im folgenden noch beschrieben werden.

c) Modellprogramme können sich untereinander nur mit Hilfe spezieller Funktionen aufrufen. Ein einfaches CALL wie z.B. CALL MP_5 ist verboten.

d) Modellprogramme dürfen keine EXIT-Statements enthalten.

e) Modellprogramme können mit dem RETURN-Statement beendet werden.

f) Neben den Modellprogrammen können weitere externe Prozeduren vorhanden sein. Sie werden nach den üblichen PL/1-

Konventionen geschrieben und aufgerufen. Sie dürfen keine PLIM-Instruktionen enthalten und nicht mit dem PLIM-Preprocessor übersetzt werden.

g) Externe Prozeduren mit dem Attribut 'MAIN' oder 'EXTERNAL' (siehe 4.2. und 4.3.) können mit Hilfe des PLIM-Preprocessors übersetzt werden.

- Prozedurkopf

Syntax:

```
Prozedurkopf          → {{Modellprozedurname:}
                         {PROC | PROCEDURE}
                         [TEST] [NUM] ; }

Modellprozedurname    →  {MP_n}

n                     →  { 1 }
                         { 2 }
                         { . }
                         { . }
                         { 99}
```

z.B.: MP_5: PROC;
MP_21:PROCEDURE TEST;

Allgemeine Regel:

a) Der Modellprozedurname ist der einzige erlaubte Entry eines Modellprogramms. Weitere Prozedurnamen bzw. Entries sind verboten.

b) Wird die TEST-Option angegeben, so bedeutet dies, daß das Programm immer im Test-Modus ablaufen kann (siehe 7.1.).

c) Bei Angabe der NUM-Option wird der vom PLIM-Preprocessor erzeugte Zwischencode stets numeriert.

- Deklarationsstatement

Syntax:

```
Deklarations-    →{{DCL | DECLARE}
statement          {[Level] Identifier [Attribut]...
                   [,[Level] Identifier [Attribut]]...
                     ;}}

Level            → übliche PL/1-Konvention

Identifier       → dürfen keine $-Zeichen enthalten,
                   sonst gelten die üblichen PL/1-
                   Konventionen

Attribut         →{[Datenattribut] Storage-Class-Attribut
                  | Program-Control-Attribut [Storage-Class-Attribut]
                  | Entry-Name-Attribut
                  | File-Description-Attribut
                  | Scope-Attribut                          }

Datenattribut    → übliche PL/1-Konvention

Storage-Class-   →{[STATIC]  }
Attribut          { CONTROLLED}
                  { BASED     }

Program-Control- →{AREA   }
Attribut          {LABEL  }
                  {OFFSET }
                  {POINTER}

Entry-Name-      →{BUILTIN}
Attribut          {ENTRY  }
                  {GENERIC}
                  {RETURNS}
```

File-Description-Attribut → übliche PL/1-Konvention

Scope-Attribut → { [INTERNAL] / EXTERNAL }

Allgemeine Regel:

a) Alle Deklarationen müssen nach dem Prozedurkopf erfolgen. Eine Ausnahme bilden die Deklarationen in Blöcken und Unterprogrammen. Für sie gibt es keine automatische Pointerverwaltung (siehe BASED-Attribut).

b) In den Deklarationen dürfen keine PLIM-Instruktionen enthalten sein.

c) Kommentar ist entsprechend den üblichen PL/1-Konventionen im Deklarationsteil erlaubt.

- Alphabetische Darstellung der Attribute

Es werden nur die Attribute beschrieben, für deren Anwendung es in Modellprogrammen bestimmte Restriktionen gibt. Zunächst gilt die Regel, alle erlaubten Attribute explizit anzugeben. Für die Verträglichkeit von Attributen unter einem Identifier gelten die üblichen PL/1-Konventionen (siehe Tabelle I-2 in / 3 /). Es folgt die Beschreibung der Attribute in alphabetischer Reihenfolge. Alternative Attribute werden jedoch zusammen behandelt.

AREA (Program-Control-Attribut)

Speicher, der mit dem AREA-Attribut definiert wird, darf nicht die Speicherklasse AUTOMATIC erhalten. Benötigt man in einem Modellprogramm viele Listen in derselben Aufruffolge, so empfiehlt es sich, diese in einer AREA anzulegen. Die zugehörigen OFFSET-Pointer können STATIC deklariert werden, ohne die Reentrant-Fähigkeit einzuschränken.

BASED, CONTROLLED, STATIC (Storage-Class-Attribut)

Die BASED-Storage-Class stellt für Modellprogramme die wichtigste Speicherklasse dar. Die Pointer der zehn ersten BASED-Deklarationen nach dem Modellprozedurnamen werden beim Verlassen einer Modellprozedur automatisch gerettet und bei Wiedereintritt zugewiesen. Damit stellt sie ein wirksames Instrument für die Reentrant-Programmierung von Modellprogrammen zur Verfügung. Die zu einer BASED-Deklaration gehörigen Pointer sollten ebenfalls explizit deklariert werden.

```
z.B.:   DCL   P     POINTER STATIC,
              A(8) CHAR(4) BASED(P);
```

Werden in Modellprogrammen mehr als zehn Identifier mit dem Attribut BASED angelegt, so müssen deren Pointer vom Benutzer selbst verwaltet werden, oder als STATIC OFFSET in einer AREA angelegt sein.

Die momentane, systemspezifische Beschränkung auf zehn BASED-Pointer ist leicht zu erweitern.

Bei der CONTROLLED- und STATIC-Speicherklasse ist besonders auf die Eindeutigkeit der zugewiesenen Werte zu achten. Ansonsten gelten die üblichen PL/1-Konventionen.

BUILTIN (Entry-Attribut)

Für die Verwaltung von Modellprogrammen werden folgende Built-In-Functions benötigt:

NULL
ADDR

Sie werden deshalb vom PLIM-Preprocessor deklariert und dürfen vom Benutzer nicht noch einmal deklariert werden.

CONTROLLED (Storage-Class-Attribut)

siehe BASED.

ENTRY und GENERIC (Entry-Name-Attribut)

Modellprogramm-Identifier mit dem Attribut ENTRY bzw. GENERIC dürfen nicht mit der Buchstabenfolge

MP ....
EV ....
MST....

beginnen. Sie sind systemspezifischen Namen vorbehalten. Dasselbe gilt auch für Deklarationen in Blöcken und Unterprogrammen.

EXTERNAL und INTERNAL (Scope-Attribut)

Die Attribute EXTERNAL und INTERNAL geben den Gültigkeitsbereich von Identifiern an. Es gelten die üblichen PL/1-Konventionen.

LABEL (Program-Control-Attribut)

Jede Label-Variable erhält vom Compiler ein zusätzliches Kennzeichen, in dem die Aktivierungsstufe eines Programms bei der Adreßzuweisung ebenfalls zugewiesen wird. Vor jeder GOTO Label-Anweisung wird die Aktualität der Label-Variablen überprüft. Für Modellprogramme ergeben sich daraus die folgenden Konsequenzen:

Steht die Wertezuweisung an eine Label-Variable und die Verzweigung mit Hilfe dieser Label-Variablen nicht innerhalb derselben PLIM-Instruktion, so führt dies zu einem Fehler. Die Deklaration von Label-Variablen wie z.B.:

```
DCL   LAB  LABEL(M1,M2,...)  STATIC;
```

ist zwar erlaubt, deren Verwendung jedoch nur innerhalb einer PLIM-Instruktion möglich. Statt dessen wird empfohlen, Label-Felder zu deklarieren und mit Hilfe von Indizes zu verzweigen.

Zum Beispiel:

```
DCL   LAB(10) LABEL STATIC,
      P       POINTER STATIC,
      I BIN FIXED BASED(P);
      I = 5;
      .
      .
      .
      GOTO LAB(I);
      .
      .
LAB(5): ;
```

OFFSET und POINTER (Program-Control-Attribut)

OFFSET- und POINTER-Variable mit der Speicherklasse AUTOMATIC verlieren am Ende jeder PLIM-Instruktion ihren Wert. Für die Deklaration und Verwendung von OFFSET- und POINTER-Variablen gelten die üblichen PL/1-Konventionen.

RETURNS (Entry-Name-Attribut)

Hier gelten die üblichen PL/1-Konventionen.

- ON-Statement

Syntax:

ON-Statement → {ON Bedingung [SNAP] On-Unit}

Bedingung → übliche PL/1-Konvention

On-Unit → { einfaches PL/1-Statement
Maschineninstruktion I
Pseudoinstruktion I
{BEGIN;} [Ausführungsstatement]
{END;} }

einfaches PL/1 Statement → siehe On-unit in / 3 /

Maschineninstruktion I
Pseudoinstruktion I
Ausführungsstatement
→ siehe PLIM-Instruktion

Allgemeine Regel:

a) Bei allen auftretenden Bedingungen, die zum Simulationsabbruch führen, kann der erreichte Zustand des Simulationslaufs mit Hilfe der $$RKDUMP-Pseudoinstruktion ausgegeben werden (siehe 7.). Die kontrollierte Beendigung des Simulationslaufes ist mit der $$STOP-Pseudoinstruktion möglich.

- PLIM-Instruktion

Syntax:

PLIM-Instruktion → {[Label:]...
Bewertungsstatement
[Ausführungsstatement]...}

Label → Labelkonstanten dürfen keine $-Zeichen enthalten, sonst gelten die üblichen PL/1-Konventionen

Bewertungsstatement → {$Bewertung␣}

Bewertung → {Bewertungskonstante | Bewertungsvariable}

Bewertungskonstante → {0 | 1 | . | . | . | 65535}

Bewertungsvariable → {Arithmetische-Variable mit den Attributen:
STATIC BIN FIXED
INITIAL (Bewertungskonstante)}

Arithmetische-Variable → Identifier mit max. 7 Zeichen (siehe Deklarationsstatement)

␣ → Blank = Leerzeichen

Beispiele für Bewertungsstatements:

a) $0
b) $100
c) DCL ADD BIN FIXED STATIC INIT(25);
   $ADD

Ausführungsstatement → {Standard-Instruktion | Gemischte-Instruktion | Maschineninstruktion I | Pseudoinstruktion I | Bibliotheksfunktion I}

Standardinstruktion → {einfaches PL/1-Statement | DO-Group (vom Typ 2 u. 3) | BEGIN-Block}

einfaches PL/1-Statement → übliche PL/1-Konvention

DO-Group (vom Typ 2 u. 3) → {DO-Statement
[Ausführungsstatement]...
END;}

BEGIN-Block → {Begin-Statement
[Deklarations-Statement]...
[On-Statement]...
[Ausführungsstatement]...
END;}

DO-Statement
Begin-Statement } → übliche PL/1-Konvention

Gemischte-Instruktion → {Ausführungsstatement
enthält:
Maschineninstruktion II
od. Pseudoinstruktion II
od. Bibliotheksfunktion II}

Maschineninstruktion I → { $$CALL
$$DISABL
$$ENABL
$$IDLE
$$JUMP
$$LOAD_P
$$LOCK
$$MASK
$$RESUME
$$RETURN
$$SAVE
$$SIGNAL
$$STOR_P
$$SVC
$$UNLOCK }

Pseudoinstruktion I → { $$INT
$$RKDUMP
$$RUN
$$STOP
$$TEST }

Bibliotheksfunktion I → { $$R_DLTE, $$R_RSET }

Maschineninstruktion II → { $$CPU_ID, $$IT_ADR, $$IT_MOT }

Pseudoinstruktion II → { $$CLOCK, $$CPUTME }

Bibliotheksfunktionen II → { $$R_CRT, $$R_SAVE, $$RANDI, $$RANDF, $$RADIS, $$BINOM, $$POISS, $$EXPON, $$ERLANG, $$GAUSS }

Beispiele für Gemischte-Instruktionen:

a) GOTO LAB($$IT_MOT);
b) PFELD($$CPU_ID)= POINTER;
c) $$INT(1,MOT,$$RANDI,P);
d) $$CALL($$IT_MOT);
e) ZEIT = $$CPUTME;

Allgemeine Regel:

a) Variablen mit der Speicherklasse AUTOMATIC verlieren am Ende einer PLIM-Instruktion ihren Wert.

b) Die Ausführung einer PLIM-Instruktion zerfällt in zwei Phasen, eine Bewertungsphase und eine Ausführungsphase. In der Bewertungsphase wird die im Bewertungsstatement

angegebene Zeit der PLIM-Maschine mitgeteilt und die Adresse des zugehörigen Ausführungsstatements festgestellt. Danach geht die Kontrolle an die PLIM-Maschine zurück. Ist die im Bewertungsstatement angegebene Zeit verstrichen, so wird das Ausführungsstatement durchlaufen.

c) Der Beginn eines neuen Bewertungsstatements kennzeichnet das Ende einer PLIM-Instruktion.

d) Da der Sprung in einen Block bzw. in eine Schleife nicht erlaubt ist, dürfen in Blöcken und Schleifen keine Bewertungsstatements enthalten sein.

e) In Do-Group vom Typ 1 sind Bewertungsstatements erlaubt, z.B.:

```
$IFB IF BITA THEN DO;
$BEWA    A = 1;
$BEWB    B = 0;
                    END;
```

f) Die Bewertung einer PLIM-Instruktion wird als Vielfaches des Grundtaktes der PLIM-Maschine angegeben.

g) Eine Programmzeile darf nur ein Bewertungsstatement enthalten.

- Unterprogramm

Syntax:

Unterprogramm → {Prozedurstatement
[Deklaration]...
[On-Statement]...
[Ausführungsstatement]...
END;}

Prozedurstatement, Deklaration → übliche PL/1-Konvention, Einschränkungen hinsichtlich der Identifier und Entry-Namen siehe Deklarationsstatement

On-Statement
Ausführungsstatement } → siehe Modellprogramm

- Prozedurende

Syntax:

Prozedurende → {[Label:]...
[Bewertungsstatement]...
END
[Modellprozedurname];}

Allgemeine Regel:

Das END des Prozedurendes darf nicht gleichzeitig das Ende einer DO-Group, eines Blocks oder das Ende einer internen Prozedur sein.

4.2. Syntaktische Beschreibung von Extern-Prozeduren, die Ausführungsstatements enthalten

Syntax:

Ext. Proz. → {{Prozedurstatement EXTERNAL;}
[Deklaration]...
[On-Statement]...
[Ausführungsstatement]...
[Unterprogramm]...
{END;}}

Prozedurstatement
Deklaration } → es gelten die üblichen PL/1-Konventionen. Einschränkungen hinsichtlich der Identifier und Entry-Namen siehe Deklarationsstatement in Kap. 4.1

On-Statement Ausführungsstatement Unterprogramm	→	siehe Modellprogramm Kap. 4.1.

Allgemeine Regel:

a) Eine Prozedur mit dem Attribut EXTERNAL wird vom PLIM-Preprocessor so verarbeitet, daß alle Maschinen- und Pseudoinstruktionen sowie die Bibliotheksfunktion benutzt werden können.

b) Externe Prozeduren dieser Art werden durch CALL aufgerufen. Sie werden als Teil einer PLIM-Instruktion behandelt, da sie keine Bewertungsstatements enthalten dürfen.

4.3. Syntaktische Beschreibung der Main-Prozedur

Syntax:

```
Main-Prozedur → {{Prozedurstatement OPTIONS(MAIN);}
                 {DCL PLIMCTL ENTRY;}
                 [Deklaration]...
                 [On-Statement]...
                 [Ausführungsstatement]...
                 {CALL PLIMCTL;}
                 [Ausführungsstatement]...
                 [Unterprogramm]...
                 {END [Prozedurname];}}
```

Prozedurstatement Deklaration On-Statement Unterprogramm Prozedurname	→	es gelten die üblichen PL/1-Konventionen. Einschränkungen hinsichtlich der Identifier und Entry-Namen siehe Kap. 4.1.

Ausführungsstatement → es dürfen keine Maschinen- und Pseudofunktionen enthalten sein. Ansonsten gelten die Bedingungen wie in Modellprogrammen.

5. Die funktionellen Komponenten des Systems

5.1. Die Eventsteuerung

Basis für die zeitliche Synchronisation aller Aktivitäten im System bilden Events. Sie werden durch eine Reihe von Unterprogrammen verwaltet und - unsichtbar für den Benutzer der PLIM - von der Maschinensteuerung aufgerufen. Die Zeitfortschaltung erfolgt auf der Basis eines Grundtaktes, dessen zeitliche Dauer frei vom Anwender interpretierbar ist.

Auf eine vertiefte Behandlung der Arbeitsweise der Eventsteuerung wird hier verzichtet, da das gesamte Unterprogrammpaket als unabhängig benutzbares System für die diskrete Eventsimulation existiert und ausführlich in / 1,2 / beschrieben ist.

5.2. Zufallszahlengenerierung

Für die Simulation der Modellumgebung werden häufig Zufallszahlengeneratoren und -transformatoren eingesetzt. Beim Aufruf liefern sie Zufallszahlen gewünschter Verteilung, die sowohl die zeitliche Distanz aufeinanderfolgender Ereignisse als auch ihre Anzahl innerhalb eines gewählten Beobachtungszeitraumes repräsentieren können.

Dementsprechend stellt das vorliegende Simulationssystem Zufallszahlengeneratoren und -transformatoren für eine kontinuierliche bzw. diskrete Verteilung der Zufallsvariablen zur Verfügung.

Die den Generierungsalgorithmen zugrundeliegenden mathematischen Verfahren, die notwendigen statistischen Tests sowie die Testergebnisse sind ausführlich in / 1 / beschrieben.

Aufgrund der dort aufgestellten Auswahlkriterien für Zufallszahlengeneratoren wurde der nach dem Verfahren von Lehmer arbeitende Zufallszahlengenerator als Basisgenerator für die Zufallszahlentransformatoren gewählt. Der Basisgenerator ist benutzertransparent. Die von ihm gelieferte ganzzahlige Basiszufallszahl dient den angeschlossenen Transformatoren als Grundlage für den Transformationsvorgang.

Die Tabelle 1 (Anhang) zeigt eine Zusammenstellung vorhandener Zufallszahlentransformatoren. Darin ist

- die Verteilung,
- der Name zugehöriger Prozedur,
- die Attribute evtl. benötigter Parameter (p1,p2), mit
  BIN FIXED(31) = FI
  BIN FLOAT = FL,
- die Bedeutung der Parameter,
- der Gültigkeitsbereich erzeugter Zufallszahlen,
- die beiden Erwartungswerte, der Mittelwert und die Varianz in Abhängigkeit von den Parametern

übersichtsmäßig enthalten.

Die Aufrufkonventionen

Die Forderung nach einer Vielzahl unabhängiger Zufallszahlentransformatoren, auch gleicher Verteilung, wurde im vorliegenden Simulationssystem durch eine - der eigentlichen Benutzung vorgeschaltete - Einrichtungsphase benutzerspezifischer Zufallszahlentransformatoren realisiert. Sie beginnt mit ihrer Erzeugung und der eindeutigen Zuordnung von Identifikationsmerkmalen zu den eingerichteten Zufallszahlentransformatoren, unter denen sie später jederzeit aufrufbar sind. Die Bedingung für eine Reproduzierbarkeit einer Folge erzeugter Zufallszahlen ist durch die zwischenzeitliche Akkumulierung der aktuellen Bezugszufallszahl gewährleistet.

Entsprechend werden Hilfsfunktionen für die Erzeugung, den Aufruf, die zwischenzeitliche Deaktivierung, die Aktivierung mit einer bestimmten Bezugszufallszahl und das Löschen von

Zufallszahlentransformatoren zur Verfügung gestellt.

Eine zusammenfassende Darstellung dieser Hilfsfunktionen in der Reihenfolge ihrer Verwendung und eine kurze Erläuterung ihrer Funktion sind in der Tabelle 2 (Anhang) enthalten. Eine detaillierte Beschreibung findet sich in / 1,2 /.

Vorgehensweise bei der Erzeugung und dem Aufruf des Zufallszahlentransformators 'name':

Für den Aufruf der Hilfsfunktion $$R_CRT (Tabelle 2) sind die Parameter sowie die Attribute der Parameter der gewünschten Verteilung 'name' notwendig. Diese findet man in der dritten Spalte der Tabelle 1. Die Funktionsprozedur $$R_CRT liefert das Identifikationsmerkmal ('pointer') des erzeugten Zufallszahlentransformators 'name'. Die Generierung einer Zufallszahl der Verteilung 'name' erfolgt nun durch den Aufruf der Hilfsfunktion $$name(pointer).

Alle Funktionen der Zufallszahlengenerierung sind unter der Bezeichnung 'Bibliotheksfunktionen' im Kap. 4.1. syntaktisch beschrieben.

5.3. Logik der PL/1-Maschine

5.3.1. Struktur der PL/1-Maschine

Die PL/1-Maschine besteht aus einer wählbaren Anzahl von Rechnerkernen, die auf einem, keinen Zugriffsbeschränkungen unterworfenen, Arbeitsspeicher unbegrenzter Größe arbeiten (siehe Abb. 5.3.-1).

Jeder Rechnerkern wird durch drei gleich strukturierte Beschreibungsblöcke repräsentiert. Sie enthalten den jeweiligen Programmstatus des

- aktuellen Programms
- zuletzt unterbrochenen Programms
- Interrupt-Programms.

Demgemäß werden drei unterschiedliche Programmzustände unterschieden:

- aktiver Status
- passiver Status
- Interrupt-Status

Jeder Status wird beschrieben durch folgende Struktur:

- Programmname
- Programmarbeitsbereich
- Ablaufpriorität
- Ablaufmodus
- - Aktion
  - Unterbrechbarkeit
  - Interrupt-Maske

Ein Rechnerkern führt immer das Programm aus, das im aktiven Status steht. Ist der Rechnerkern unterbrechbar und nicht maskiert, so führt ein Interrupt zu einem Statuswechsel im Rechnerkern. Der aktive Status ersetzt den passiven, der Interrupt-Status wird in den aktiven Status kopiert und damit das ablaufbereite Interrupt-Programm initialisiert.

Jeder in der PL/1-Maschine erzeugte Interrupt wird dargestellt durch ein Interrupt-Wort.

Das Interrupt-Wort beinhaltet:

- Verkettungszeiger
- Parameterablageadresse
- Interrupt-Schlüssel
- Interrupt-Priorität

Interrupts an gleichen Rechnerkernen werden prioritätsgeordnet aneinandergekettet. Der Rechnerkern hat unmittelbaren Zugriff auf das aktuelle, d.h. zum letzten Interrupt gehörige Interrupt-Wort.

Für eine eindeutige Identifikation sind die Rechnerkerne mit Identifikationsnummern versehen und ihre Beschreibungen untereinander verkettet.

Abb. 5.3.-2 zeigt die Organisation eines Rechnerkerns im Überblick.

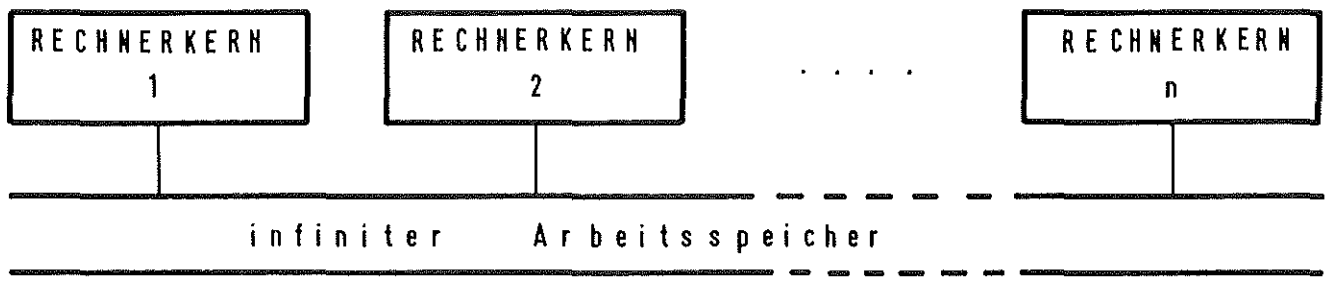


Abb.: 5.3-1 Konfiguration der PL/1-Maschine

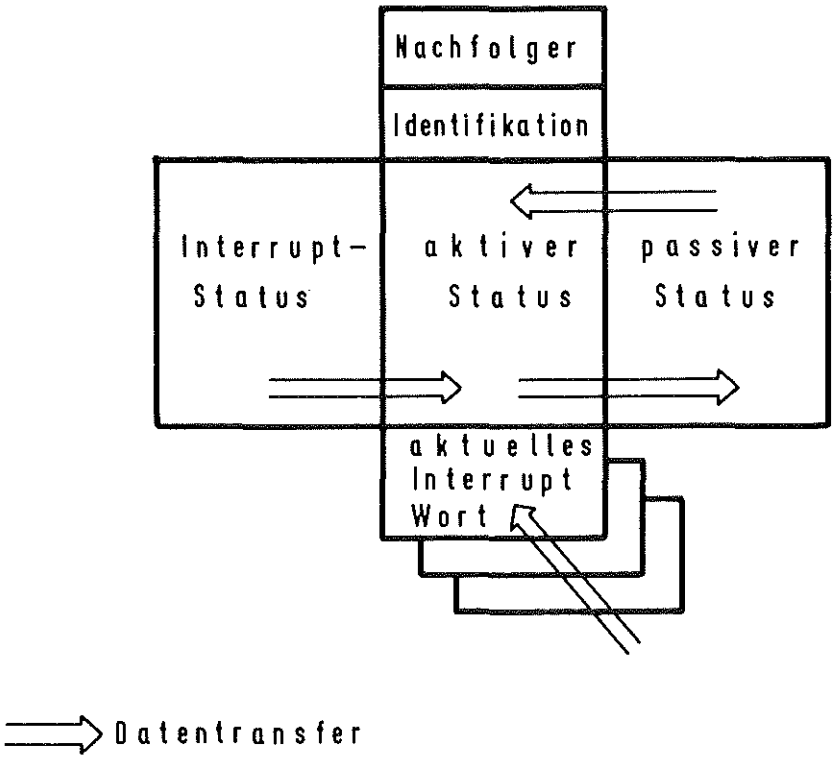


Abb.: 5.3-2 Struktur eines Rechnerkerns

5.3.2. Datenstruktur eines Rechnerkerns

Ein Rechnerkern wird durch folgende PL/1-Datenstruktur beschrieben:

```
1 RK,
  2 NACHF POINTER,
  2 ID_RK BIN FIXED(31),
  2 ST_AKT,
    3 PROG_NAME BIN FIXED(31),
    3 PROG_BER  POINTER,
    3 PRIO      BIN FIXED,
    3 MODUS,
      4 AKTION    BIN FIXED,
      4 IT_SPERRE BIN FIXED,
      4 IT_MASKE  BIN FIXED,
  2 ST_IT  LIKE ST_AKT,
  2 ST_RET LIKE ST_AKT,
  2 IT,
    3 PUFFER POINTER,
    3 ADR    POINTER,
    3 MOTIV  BIN FIXED,
    3 PRIO   BIN FIXED;
```

Erläuterung der Variablen:

- NACHF

  Zeiger, der auf die Datenstruktur des nächsten Rechnerkerns zeigt.

- ID_RK

  eine den Rechnerkern bezeichnende Identifikationsnummer, die dem Benutzer durch die Maschinen-Instruktion $$CPU_ID mitgeteilt wird.

- ST_AKT

  Unterstruktur: aktiver Status.

- PROG_NAME

  Laufende Nummer des aktiven Modellprogramms.

- PROG_BER

  Zeiger zum Datenbereich des aktiven Modellprogramms. Der Datenbereich wird jeweils bei Programminitialisierung vom System allokiert.

- PRIO

  Priorität, unter der der Rechnerkern abläuft. Sie wird vom Anwender angegeben. (Es gilt: wachsende Zahl = fallende Priorität, höchste Priorität = 1.)

- MODUS

  Unterstruktur: Ablaufmodus des Programms.

- AKTION

  Nummer, die die Ablaufart des Rechnerkerns kennzeichnet. Unterschieden wird zwischen:

  1: Aktiv, der Rechnerkern arbeitet die PLIM-Instruktionsfolge eines Programms ab.

  2: Kreisend, der Rechnerkern befindet sich in der Ausführung einer $$LOCK-Instruktion.

  3: Pseudo-Aktiv, der Rechnerkern befindet sich in einer simulierten Ausführungsphase, die durch die Pseudo-Instruktion $$RUN eingeleitet wurde.

  4: Idle, der Rechnerkern führt keine Instruktionsfolge aus. Dieser Zustand wird bewirkt durch die Maschinen-Instruktion $$IDLE.

- IT_SPERRE

  Kennzahl, die angibt, ob ein Interrupt sofort verarbeitet oder gekellert wird. Es gilt

  0: Programm ist ununterbrechbar (disabled).

  1: Programm ist unterbrechbar (enabled).

- IT_MASKE
  Interrupt-Maske mit folgendem Schlüssel:

  0: unmaskiert, Interrupt wird angenommen und in die Interruptkette eingeordnet.

  1: maskiert, Interrupt wird ignoriert.

- ST_IT
  Unterstruktur: Interrupt-Status. Die Struktur ist gleich der Unterstruktur ST_AKT.

- PROG_NAME und PRIO werden bei Rechnerkernkonfigurierung vom Benutzer angegeben.

- ST_RET
  Unterstruktur: Passiver Status. Die Struktur ist gleich der Unterstruktur ST_AKT.

- IT
  Unterstruktur: Beschreibung des zum zuletzt akzeptierten Interrupt gehörigen Interrupt-Wortes.

- PUFFER
  Zeiger zum prioritätshöchsten anstehenden Interrupt-Wort.

- ADR
  Zeiger zu einem Bereich, in dem zum aktuellen Interrupt zugehörige Information abgelegt ist. Die Anfangsadresse wird dem Anwender durch die Maschineninstruktion $$IT_ADR mitgeteilt.

- MOTIV
  Ein Schlüssel, der den aktuellen Interrupt spezifiziert. Die Angabe erfolgt mittels der Maschinen-Instruktion $$IT_MOT.

Folgende Werte sind reserviert:

1: Schlüssel des Programmende-Interrupts (s. 5.3.4.)

2: Schlüssel des Konsolstart-Interrupts (s. 5.3.4.)

3: SVC-Schlüssel

- PRIO

  Priorität, nach der das Interrupt-Wort in die Interrupt-Kette eingeordnet wurde. (Es gilt: wachsende Zahl = fallende Priorität.)

5.3.3. Instruktionsvorrat der PL/1-Maschine

Die PL/1-Maschine enthält neben den üblichen PL/1-Statements zwei zusätzliche Klassen von Instruktionen:

- Maschineninstruktionen und
- Pseudoinstruktionen

Der syntaktische Aufbau der Instruktionen ist ausführlich in 4.1. beschrieben.

Maschineninstruktionen

Die Maschineninstruktionen dienen zur Manipulation der Rechnerkerne, zur Kommunikation und Synchronisation zwischen Rechnerkernen und zur Steuerung kontrollierter Übergänge zwischen Modellprogrammen.

Folgende Maschineninstruktionen stehen zur Verfügung:

$$DISABL	$$SIGNAL	$$RETURN
$$ENABL	$$SVC	$$RESUME
$$MASK	$$IT_ADR	$$CALL
$$IDLE	$$IT_MOT	$$STOR_P
$$CPU_ID	$$LOCK	$$LOAD_P
$$SAVE	$$UNLOCK	$$JUMP

Maschinen-Instruktionen zur Steuerung und Manipulation von Rechnerkernen:

- $$ENABL

  Syntax: keine Parameter
  Funktion: Der Rechnerkern wird in den Zustand 'unterbrechbar' gesetzt. (ST_AKT.IT_SPERRE = 1)

- $$DISABL

  Syntax: keine Parameter
  Funktion: Der Rechnerkern wird in den Zustand 'ununterbrechbar' gesetzt.
  (ST_AKT.IT_SPERRE = 0)

- $$MASK

  Syntax: $$MASK (maske);
  maske: BIN FIXED;
  Funktion: Eine Interrupt-Maske wird gesetzt (ST_AKT.MASKE = maske),
  es gilt:
  maske = 1: jeder auftretende Interrupt wird ignoriert
  maske = 0: ein auftretender Interrupt wird in die Interrupt-Kette eingereiht.

  (maske ≠ 1 gleichbedeutend wie maske = 0)

- $$IDLE

  Syntax: keine Parameter
  Funktion: Der Rechnerkern beendet das Abarbeiten von PLIM-Instruktionen und wird 'idle'. Die Instruktion impliziert das Lösen der Interruptsperre (ST_AKT.IT_SPERRE = 1) und die Demaskierung des Interrupteingangs (ST_AKT.IT_MASKE = 0). Die Reaktivierung des Rechnerkerns ist nur mittels eines Interrupts möglich. Ein anstehender oder eintreffender Interrupt führt zur Initialisierung des Interrupt-Programms. Dabei entfällt das sonst übliche vorher-

gehende Überschreiben des passiven durch den aktiven Status.

- $$SAVE

  Syntax: $$SAVE(adr);
  adr: POINTER;

  Funktion: Der passive Status wird auf den 'adr' folgenden Speicherplätzen abgelegt.
  'adr' muß Anfangsadresse folgender Struktur sein:

```
1 PASS_RK,
  2 PROG_NAME BIN FIXED(31),
  2 PROG_BER  POINTER,
  2 PRIO      BIN FIXED,
  2 MODUS,
    3 AKTION    BIN FIXED,
    3 IT_SPERRE BIN FIXED,
    3 IT_MASKE  BIN FIXED;
```

  Die Instruktion ermöglicht dem Anwender, verdrängte Programme selbst zu verwalten. Die Programme können mit der Komplementär-Instruktion $$RESUME wieder aktiviert werden.

- $$CPU_ID

  Syntax: keine Parameter, die Instruktion ist wie eine Funktionsprozedur zu benutzen, der übergebene Wert ist vom Typ 'BIN FIXED'.

  Funktion: Die Identifikationsnummer des Rechnerkerns wird mitgeteilt.

Maschinen-Instruktionen zur Kommunikation und Synchronisation zwischen Rechnerkernen:

- $$SIGNAL

  Syntax: $$SIGNAL(rknr,motiv,prio,adr);

```
rknr:  BIN FIXED,
motiv: BIN FIXED,
prio:  BIN FIXED,
adr:   POINTER;
```

Funktion: Ein Interrupt wird an den Rechnerkern mit der Nummer 'rknr' geschickt. Ist der Rechnerkern unmaskiert, so wird ein Interruptwort gebildet und in die Interruptkette des spezifizierten Rechnerkerns eingehängt.

Das Interruptwort setzt sich aus folgenden Komponenten zusammen:

- motiv: eine frei wählbare Zahl, die etwa als Interrupt-Schlüssel benutzt werden kann.
- prio: eine dem Interrupt zugeordnete Priorität, nach der das Interruptwort in die Interruptkette des Rechnerkerns 'rknr' eingeordnet wird (wachsende Zahl = fallende Priorität, Ordnungskriterium bei gleichen Prioritäten: FIFO).
- adr: Anfangsadresse eines vom Anwender allokierten Bereichs zur Ablage interruptspezifischer Information.

Ist der adressierte Rechnerkern maskiert, so wird kein Interruptwort kreiert.

Wird ein Interrupt an einen nicht existierenden Rechnerkern geschickt, so erfolgt die Ausgabe einer entsprechenden Meldung und mit Hilfe der $$STOP-Instruktion der Abbruch des Simulationslaufes.

- $$SVC

Syntax:
```
$$SVC(adr);
adr: POINTER;
```

Funktion: Die Instruktion $$SVC(adr) stellt eine normierte Kurzform der Instruktion

```
$$SIGNAL($$CPU_ID,3,1,adr)
```

dar.

- $$IT_ADR

  Syntax: kein Parameter, die Instruktion ist wie eine Funktionsprozedur zu benutzen. Der übergebene Wert ist vom Typ 'POINTER'.

  Funktion: Die Instruktion liefert die im aktuellen Interruptwort abgelegte Anfangsadresse der Interrupt-Information.

- $$IT_MOT

  Syntax: keine Parameter, die Instruktion ist wie eine Funktionsprozedur zu benutzen. Der übergebene Wert ist vom Typ 'BIN FIXED'.

  Funktion: Die Instruktion gibt den Wert des im aktuellen Interruptwort spezifizierten Motivs (Interrupt-Schlüssel) an.

- $$LOCK

  Syntax:
  ```
  $$LOCK(v);
  v: POINTER;
  ```

  Funktion: Mit der Instruktion $$LOCK kann die Zuteilung eines nur seriell benutzbaren Betriebsmittels bei gleichzeitiger Anforderung von verschiedenen Rechnerkernen geregelt werden. Ein Betriebsmittel wird durch einen definierten Wert der Lockvariablen 'v' beschrieben (z.B.: v = addr (Betriebsmittel)). Derjenige Rechnerkern, der die Lockvariable setzt, bekommt das Betriebsmittel exklusiv zugeteilt und kann in der Abarbeitung der Instruktionsfolge fortfahren / 5 /.

  Die Instruktion $$UNLOCK(v) beendet das Benutzungsrecht auf das Betriebsmittel v, d.h. die Lockvariable wird frei.

Die Instruktion $$LOCK(v) bewirkt im Fall

- Lockvariable v nicht gesetzt:
  Setzen der Lockvariablen v
- Lockvariable v gesetzt:
  Kreisen des Rechnerkerns auf der $$LOCK-Instruktion (ST_AKT.AKTION=2) solange, bis die Lockvariable v frei wird. (Der Rechnerkern gilt währenddessen als ununterbrechbar, die Wartezeit wird der CPU-Zeit zugerechnet.)

Ist die Lockvariable v frei, und mehrere Rechnerkerne führen gleichzeitig die $$LOCK-Instruktion aus, so gilt im Fall

- Lockvariable v nicht gesetzt:
  derjenige Rechnerkern setzt die Lockvariable v, der unter der höchsten Priorität abläuft
- Lockvariable v war gesetzt und wird frei:
  aufgrund einer Zufallsentscheidung wird einer der kreisenden Rechnerkerne berechtigt, die Lockvariable v zu setzen.

- $$UNLOCK

  Syntax: $$UNLOCK(v);
  v: POINTER;

  Funktion: Die Instruktion $$UNLOCK beendet das exklusive Benutzungsrecht des durch 'v' gekennzeichneten Betriebsmittels - die Lockvariable v wird frei. $$UNLOCK ist die Komplementärinstruktion zu $$LOCK.

Maschineninstruktionen zur Lenkung kontrollierter Übergänge zwischen Modellprogrammen:

- $$RETURN

  Syntax: keine Parameter

Funktion: Die Instruktion veranlaßt ein Überschreiben des aktiven Status durch den passiven Status.

- $$RESUME

Syntax:
```
$$RESUME(adr);
adr: POINTER;
```

Funktion: Die Instruktion $$RESUME stellt zusammen mit der Instruktion $$SAVE ein wichtiges Werkzeug zur kontrollierten Ablaufverwaltung von Modellprogrammen dar.
Die Instruktion $$RESUME(adr) überschreibt den aktiven Status mit der bei 'adr' abgelegten Struktur.
Der Pointer 'adr' muß auf eine Struktur mit folgendem Aufbau zeigen (s. $$SAVE):

```
1 RK_STR,
  2 PROG_NAME BIN FIXED(31),
  2 PROG_BER POINTER,
  2 PRIO     BIN FIXED,
  2 MODUS,
    3 AKTION    BIN FIXED,
    3 IT_SPERRE BIN FIXED,
    3 IT_MASKE  BIN FIXED;
```

Die Instruktion wird verwendet:

- zur Aktivierung von verdrängten Programmen, die mit ihrem Status aufgrund der Instruktion $$SAVE und 'adr' abgelegt worden sind;
- zur Initialisierung von Programmen, wobei folgende Regeln vorgeschrieben sind:

  PROG_NAME: Nummer des zu initialisierenden Programms
  PRIO: Ablaufpriorität > 0
  AKTION: 1
  PROG_BER: NULL

IT_SPERRE: 0 falls ununterbrechbar
1 falls unterbrechbar
IT_MASKE: 0 falls unmaskiert
1 falls maskiert

Werden diese Regeln nicht eingehalten, so wird eine entsprechende Meldung ausgegeben und der Simulationslauf abgebrochen.

- $$CALL

Syntax: $$CALL (prognr);
prognr: BIN FIXED(31);

Funktion: Die Instruktion $$CALL (prognr) dient zum Unterprogrammaufruf des durch 'prognr' gekennzeichneten Modellprogramms. Nach Ablauf des spezifizierten Programms wird die Kontrolle an das aufrufende Programm zurückgegeben.
Die Parameterübergabe wird durch die Instruktionen $$LOAD_P und $$STOR_P gesteuert.
Die Reentrantfähigkeit der Modellprogramme ermöglicht eine rekursive Benutzung der Instruktion $$CALL.
Die Angabe einer nicht existierenden Programmnummer führt zum Abbruch der Simulation.

- $$STOR_P

Syntax: $$STOR_P (parmadr);
parmadr: POINTER;

Funktion: Die Instruktion $$STOR_P dient zur Parameterübertragung an ein durch $$CALL aufzurufendes Unterprogramm.
Der Pointer 'parmadr' weist auf einen vom Anwender allokierten Bereich, der die Parameter enthält. Diese Adresse wird im Programmarbeitsbereich abgelegt und ist mit Hilfe der Instruktion $$LOAD_P im aufgerufenen Programm verfügbar.

Wird die Instruktion $$STOR_P in einem Programm mehrfach verwendet, so wird jeweils die zuletzt abgelegte Adresse überschrieben.

- $$LOAD_P

Syntax: $$LOAD_P (parmadr);
parmadr: POINTER;

Funktion: Durch die Instruktion wird dem Pointer 'parmadr' die Adresse zugewiesen, die im aufrufenden Programm mit Hilfe der Instruktion $$STOR_P abgelegt wurde.
Der Pointer 'parmadr' erhält den Wert NULL, wenn
- im aufrufenden Programm keine Parameteradresse abgelegt wurde
- das Programm nicht aufgrund der Instruktion $$CALL aktiviert wurde.

- $$JUMP

Syntax: $$JUMP (prognr);
prognr: BIN FIXED(31);

Funktion: Die Instruktion $$JUMP bewirkt die Aktivierung des Modellprogramms mit der Nummer 'prognr'. Im Gegensatz zur Instruktion $$CALL erfolgt nach Ablauf des aktivierten Modellprogramms keine Rückgabe der Ablaufkontrolle.
Eine Parameterübergabe mit Hilfe von Maschinen-Instruktionen ist nicht möglich.

Pseudoinstruktionen

Die Pseudoinstruktionen sind Hilfsmittel zur effizienten Handhabung der PL/1-Maschine als ein Simulationsinstrument. Sie bilden ein Werkzeug zur

- Simulation externer Interrupts: $$INT
- Simulation von Rechenzeiten: $$RUN
- Zeitnahme: $$CLOCK, $$CPUTME
- Überwachung der Ablauffolge der PLIM-Instruktionen: $$TEST
- Ausgabe der Rechnerkernzustände: $$RKDUMP

Beschreibung der Pseudoinstruktionen:

- $$INT

  Syntax:
  ```
  $$INT (zeit,motiv,prio,adr);
  zeit:  BIN FIXED(31),
  motiv: BIN FIXED,
  prio:  BIN FIXED,
  adr:   POINTER;
  ```
  Funktion: Die Instruktion erzeugt nach Ablauf der Zeit 'zeit' einen Interrupt am eigenen Rechnerkern. $$INT (zeit,motiv,prio,adr) impliziert zum spezifizierten Zeitpunkt die oben beschriebene Maschineninstruktion $$SIGNAL($$CPU_ID,motiv,prio,adr)

- $$RUN

  Syntax:
  ```
  $$RUN (zeit);
  zeit: BIN FIXED(31);
  ```
  Funktion: Die Instruktion $$RUN (zeit) dient zur Simulation der Rechenzeit 'zeit'. Für die Dauer dieser Zeitspanne wird der Rechnerkern in den pseudo-aktiv Zustand versetzt (ST_AKT.AKTION=3). Der Rechnerkern kann innerhalb dieser Phase zu jedem Zeitpunkt unterbrochen werden. Entsprechend der Restzeit wird bei Reaktivierung die Ausführung der RUN-Phase festgesetzt.

- $$CPUTME

Syntax: keine Parameter, die Instruktion ist wie eine Funktionsprozedur zu benutzen. Der übergebene Wert ist vom Typ 'BIN FIXED(31)'.

Funktion: Die seit Aktivierung der aktuellen Programmfolge aufgelaufene Rechenzeit wird geliefert. (Eine geschlossene Programmfolge beginnt mit einem Interrupt-Programm oder einem mittels $$RESUME gestarteten Modellprogramm und umfaßt alle durch $$CALL oder $$JUMP angeschlossenen Modellprogramme.)Die angegebene CPU-Zeit schließt die Ausführungszeit der momentanen PLIM-Instruktion mit ein.
Neben der Aktiv-Zeit werden zur CPU-Zeit die Zeitintervalle addiert, in denen sich die Programmfolge in $$RUN- oder $$LOCK-Phasen befand. $$IDLE-Phasen und Phasen in denen die Programmfolge nicht im aktiven Zustand war, werden nicht gezählt.

- $$CLOCK

Syntax: keine Parameter, die Instruktion ist wie eine Funktionsprozedur zu benutzen. Der übergebene Wert ist vom Typ 'BIN FIXED(31)'.

Funktion: Die Instruktion liefert die aktuelle Simulationszeit, d.h. die Anzahl der seit Simulationsbeginn verstrichenen Grundtakte. Der übergebene Wert stellt die Simulationszeit nach Ausführung der momentanen PLIM-Instruktion dar.

- $$TEST

Syntax:
```
$$TEST(z);
z: CHAR(1);
```

Funktion: Mit dem Aufruf der Instruktion $$TEST('B') beginnt eine Testphase, die die Ablaufkontrolle aller aktiven Modellprogramme ermöglicht. Die Instruktion $$TEST('E') beendet die Testphase. Testphasen können sich zeitlich überlagern. Während einer Testphase werden vor Ausführungsbeginn einer PLIM-Instruktion folgende Werte ausgegeben:

CLOCK: Simulationszeit, bezogen auf Ausführungsende der PLIM-Instruktion (s. $$CLOCK)

RK_NR: Nummer des aktiven Rechnerkerns

PROG_N: Nummer des aktiven Modellprogramms

INDEX: Aktivierungsindex des Modellprogramms (Kennzahl des Programmbereichs als Unterscheidungsmerkmal paralleler Programmbenutzung)

STMT: Statementnummer der auszuführenden PLIM-Instruktion (Jede PLIM-Instruktion wird bei Übersetzung mit der TEST-Option statisch durchnumeriert. Die erste Bewertung impliziert die Statementnummer 1. Die höchste Statementnummer ist gleich der Anzahl der Bewertungen. Erfolgte die Übersetzung ohne Test-Option, so gilt: STMT=0.)

CPUTME: Die CPU-Zeit einschließlich der Ausführungszeit der vorliegenden PLIM-Instruktion (s. $$CPUTME)

Gilt z ≠ 'B' oder z ≠ 'E' so wird eine entsprechende Meldung ausgegeben - die Simulation wird fortgesetzt.

- $$RKDUMP

Syntax: keine Parameter
Funktion: Die Instruktion liefert den augenblicklichen Inhalt der Rechnerkerne. In der CPU-Zeit der jeweils aktiven Programmfolge ist die Ausführungszeit der momentanen PLIM-Instruktion enthalten.

5.3.4. Organisatorischer Aufbau des Programmsystems

Ablaufsteuerung

Die interne Ablaufsteuerung der PL/1-Maschine geschieht mittels dreier Event-Prozeduren, die unter der Regie der Eventsteuerung (s. 5.1.) ablaufen.

Die Synchronisation und zeitliche Fortschaltung der Abläufe in der PL/1-Maschine wird durch Events geregelt. Mit ihrer Hilfe wird die Auflösung paralleler Aktivitäten in sequentielle Einzelschritte vorgenommen.

In der Anordnung der Events spiegelt sich die Abarbeitungsfolge der PLIM-Instruktionen wieder. Die Aktivierung der Rechnerkerne und damit die Reihenfolge der Events auf der Zeitachse ist nach folgendem Prinzip geregelt.

Es sei

zp(i,k): absoluter Simulationszeitpunkt, an dem der Rechnerkern i die PLIM-Instruktion k ausgeführt hat. (Programmteile eines Modellprogramms, die vor Erreichen des ersten Bewertungszeichens durchlaufen wurden (k=0), zählen zur Initialisierung und damit zur Ausführungszeit der zuletzt abgearbeiteten PLIM-Instruktion)

pr(i): Priorität, unter der der Rechnerkern i abläuft.

Unter den Zeitpunkten zp sei folgende Relation '$\lessdot$' definiert:

zp(s,t) $\lessdot$ zp(u,v) genau dann, wenn

1.) zp(s,t) < zp(u,v)

oder

2.) zp(s,t) = zp(u,v) und
    pr(s) < pr(u).

oder

3.) zp(s,t) = zp(u,v) und
    pr(s) = pr(u) und
    zp(s,t-1) $\lessdot$ zp(u,v-1)

Es gilt folgende Regel:

Als nächster wird der Rechnerkern j zur Ausführung der Instruktion l aktiviert, für den gilt:

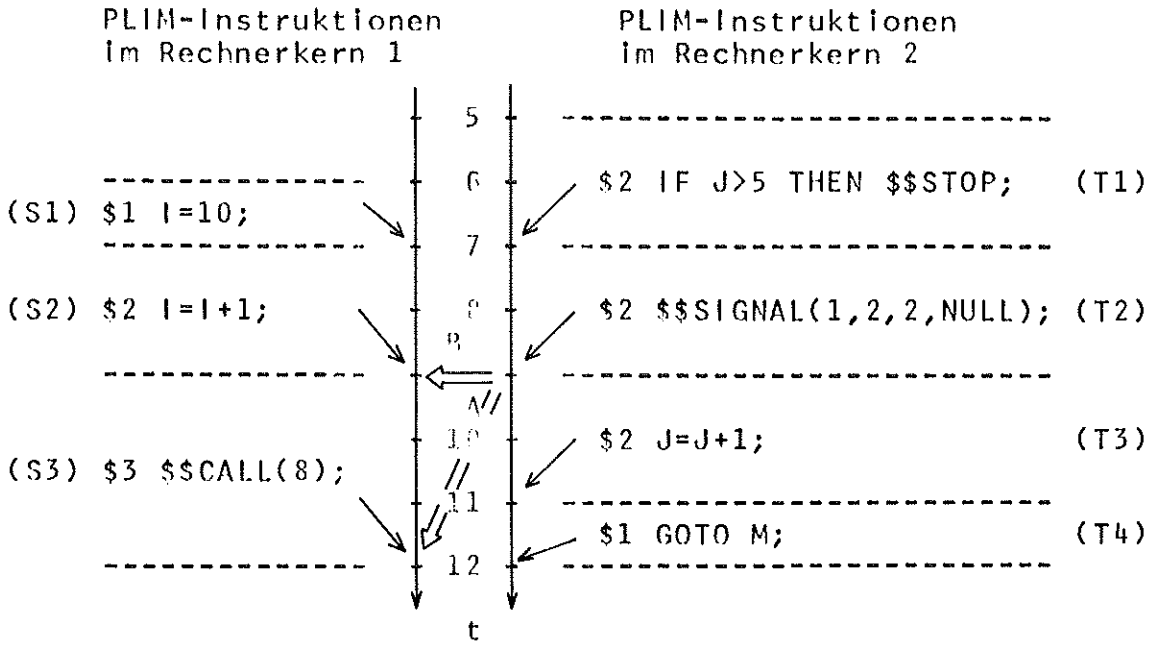
zp(j,l) $\lessdot$ zp(i,k) für alle i,k (i$\neq$j, k$\neq$l)

Diese Regel gewährleistet bei Abbildung der Zeitpunkte in Events die geforderte zeitgerechte Aktivierung der Event-Prozeduren und damit das gewünschte Verhalten der PLIM-Maschine. Abb. 5.3.-3 soll den Vorgang der Serialisierung paralleler Abläufe anhand eines Beispiels verdeutlichen.

Die Event-Prozedur MST_1

Die Aktivierung der Event-Prozedur wird ausgelöst bei Eintritt eines Events mit folgender Datenstruktur:

TIME = zp(j,l)
PRIO = Ablaufpriorität des Rechnerkerns j



⟶ Zeitpunkt der realen Ausfuehrung der PLIM-Instruktion

⟹ Zeitpunkt der Initialisierung des Interrupt-Programms im Rechnerkern 1

Fall A : Prioritaet Rechnerkern 1 > Prioritaet Rechnerkern 2

Fall B : Prioritaet Rechnerkern 1 < Prioritaet Rechnerkern 2

Serialisierte Ablauffolge der PLIM-Instruktionen im

Fall A : S1 / T1 / S2 / T2 / T3 / S3 / T4 / Beginn Interrupt-Programm im Rechnenrkern 1

Fall B : T1 / S1 / T2 / S2 / ... Beginn Interrupt-Programm im Rechnerkern 1

Abb.: 5.3-3 Serialisierung parallelablaufender PLIM-Instruktionen anhand eines Beispiels

ENTRY = 1 (MST_1)
SV_AREA = Pointer zur Datenstruktur des Rechnerkerns j

Über die Datenstruktur des Rechnerkerns hat die Prozedur Zugriff auf die Save-Area des zu aktivierenden Modellprogramms und kann somit über die gesamte Information von Programm und Rechnerkern verfügen.

Die Prozedur bildet die Funktion des Befehlswerkes nach, indem sie die Ausführung einer PLIM-Instruktion durch Aufruf des Modellprogramms initialisiert.

Nach Ausführung der PLIM-Instruktion erfolgt eine ablaufspezifische Auswertung, wie Reaktion auf anstehende Interrupts, Behandlung von Programmende oder Disposition bei geändertem Rechnerkernzustand.

Wird das - die aktuelle Programmfolge abschließende - Programmende erreicht, so löst die Prozedur die Interrupt-Sperre, demaskiert den Interrupt-Eingang und kreiert folgenden Programmende-Interrupt:

MOTIV = 1
ADR = NULL
PRIO = 1

Abhängig von den getroffenen Aktionen wird das Event unter Berücksichtigung der Ausführungszeit der folgenden PLIM-Instruktion neu aufgesetzt.

Durch diesen Mechanismus wird für die Reaktivierung des Rechnerkerns und Weiterbehandlung des Programms gesorgt.

Die Event-Prozedur MST_2

Das die Prozedur aktivierende Event wird von der Pseudoinstruktion $$STOP aufgesetzt. Es enthält folgende Datenstruktur:

TIME = 0
PRIO = 0
ENTRY = 2 (MST_2)
SV_AREA = NULL

Die Prozedur führt einen kontrollierten Simulationsabbruch herbei, indem sie alle bestehenden Events löscht.

Die Event-Prozedur MST_3

Die Prozedur sorgt für die zeitgerechte Generierung eines von der Pseudoinstruktion $$INT abgesetzten Interrupts.

Das zugehörige Event hat folgende Datenstruktur:

TIME = Zeit des Interrupt-Eintrittes
PRIO = 0
ENTRY = 3 (MST_3)
SV_AREA = Zeiger zur Parameterliste von $$INT

Bei Eintritt des Events wird die Maschineninstruktion $$SIGNAL mit den in der Parameterliste spezifizierten Werten aufgerufen.

Rechnerkernkonfigurierung und Start der PL/1-Maschine

Zur Rechnerkernkonfigurierung und Start der PL/1-Maschine muß die Prozedur PLIMCTL vom anwendereigenen Hauptprogramm aufgerufen werden.

PLIMCTL umschließt den Simulationslauf und gibt nach Simulationsende die Kontrolle an das Hauptprogramm zurück (s. 7.1.).

Entsprechend den Benutzerangaben allokiert die Prozedur die Datenstrukturen für die Rechnerkerne und ordnet ihnen ein Interrupt-Programm einschließlich Ablaufpriorität zu (s. 8.1.).

Der aktive und passive Status wird auf folgende Werte normiert:

PROG_NAME = 0
PROG_BER = NULL
PRIO = 1
AKTION = 4 (idle)
IT_SPERRE = 1 (enable)
IT_MASKE = 0 (unmaskiert)

Der Interrupt-Status erhält folgende Werte:

PROG_NAME = Modellprogrammnummer des zugeordneten Interrupt-Programms
PROG_BER = NULL
PRIO = angegebene Ablaufpriorität (> 0)
AKTION = 1 (aktiv)
IT_SPERRE = 0 (disable)
IT_MASKE = 0 (unmaskiert)

Den Rechnerkernen werden in der Reihenfolge der Allokation von 1 an steigende Identifikationsnummern zugeordnet.

Nach der Konfigurierung wird der Anwender aufgefordert, die zu aktivierenden Rechnerkerne anzugeben (s. 8.1.). Die Prozedur kreiert für jeden angegebenen Rechnerkern einen Konsolstart-Interrupt mit folgenden Parametern:

MOTIV = 2
ADR = NULL
PRIO = 0

Damit werden die angesprochenen Rechnerkerne 'geweckt' und beginnen zur Simulationszeit 0 mit der Abarbeitung der zugeordneten Interrupt-Programme.

In der Realisierung läuft zur Zeit 0 die Reihenfolge der Rechnerkernaktivierungen geordnet nach den Ablaufprioritäten der Interrupt-Programme ab (bei Gleichheit werden Rechnerkerne mit kleineren Identifikationsnummern vorgezogen).

Aufbau des Programmsystems

Das gesamte Programmsystem ist hierarchisch gegliedert. Die einzelnen Schichten bestehen aus externen Prozeduren, die sich in folgende vier Klassen einteilen lassen:

- Prozeduren der Eventsteuerung
- Prozeduren der PL/1-Maschine
- Prozeduren der Zufallszahlengenerierung
- Prozeduren des Anwenders.

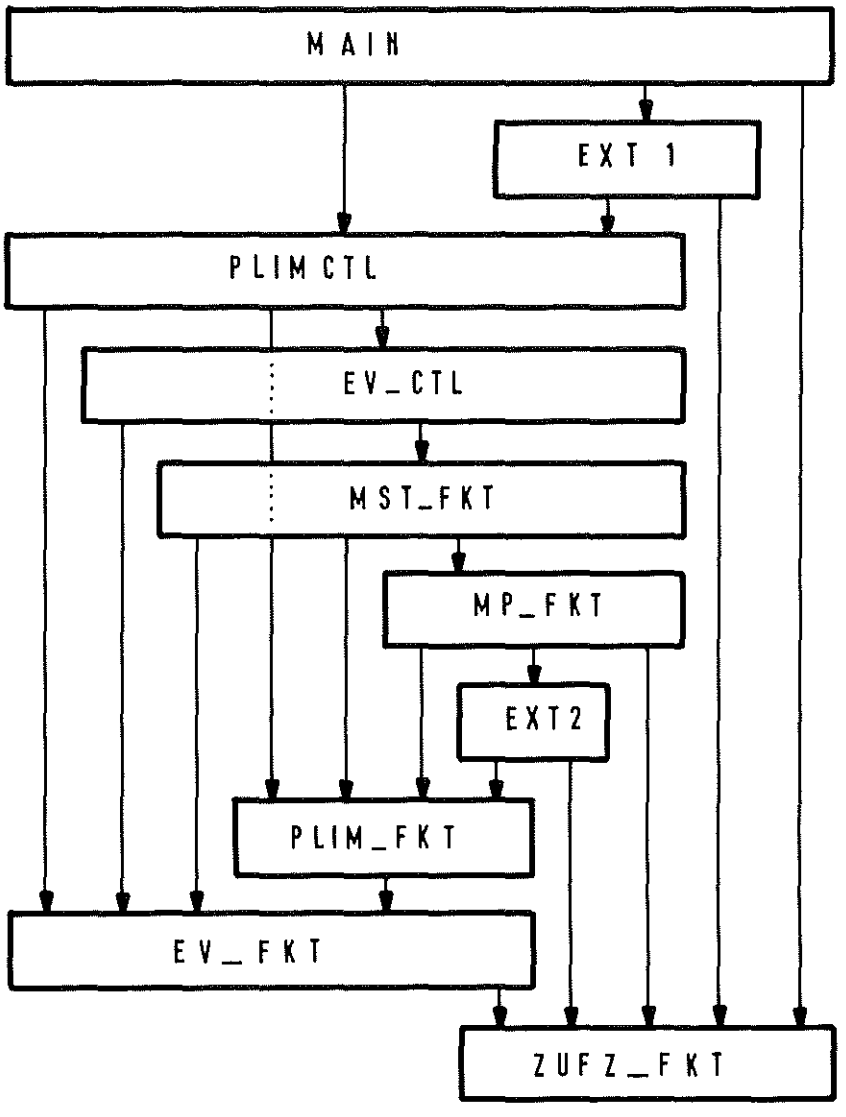
Abb. 5.3.-4 zeigt den strukturellen Aufbau des gesamten Programmsystems. Die Aufrufmächtigkeit der einzelnen Schichten wird im Bild durch Verbindungen verdeutlicht.

5.4. Preprocessor

Modellprogramme werden zur Laufzeit nicht interpretiert, sondern unter Kontrolle der PL/1-Maschine instruktionsweise durchlaufen (Kap.3). Die dazu erforderliche Aufbereitung des Quellcodes erfolgt mittels eines Preprocessors. Ein derart modifiziertes Programm ist außerhalb des Simulationssystems nicht mehr ausführbar. Daraus ergibt sich die Forderung nach einem Preprocessor für den offline Test von Modellprogrammen und einem Preprocessor, der Modellprogramme so modifiziert, daß sie unter Kontrolle der PL/1-Maschine ausgeführt werden können. Es folgt eine Funktionsbeschreibung beider Preprocessoren.

5.4.1. TSTSCAN - ein Preprocessor für den Offline-Test von Modellprogrammen

TSTSCAN reduziert den Quellcode eines Modellprogramms in ein gültiges PL/1-Programm und unterzieht es einer Syntaxprüfung. Dazu führt es folgende Modifikationen durch:



MAIN - Hauptprogramm
EXT1 - Externe Prozeduren
MP_FKT - Modellprogramme
EXT2 - Externe Prozeduren

} Prozeduren des Benutzers

PLIMCTL - Initialisierungsprozedur
MST_FKT - Eventprozeduren
PLIM_FKT - Maschinen-und Pseudoinstruktionen
EV_CTL - Eventkontrolle
EV_FKT - Prozeduren der Eventsteuerung
ZUFZ_FKT - Bibliotheksfunktionen

} Prozeduren des Simulationssystems .

Abb.: 5.3–4 Organisatorischer Aufbau des Programmsystems

- In der PROC-Anweisung des Prozedurkopfes wird 'OPTIONS (MAIN)' eingefügt
- Alle in der Prozedur enthaltenen Bewertungsstatements, Maschineninstruktionen I, Pseudoinstruktionen I und Bibliotheksfunktionen I werden in Kommentarzeichen (/+ ..... +/) eingeschlossen und dadurch unwirksam gemacht.
- In die Modellprozedur werden die Deklaration 'DCL NULL BUILTIN;' und weitere Deklarationen eingefügt, so daß die Modellprozeduren einschließlich der noch wirksamen PLIM-Funktionen von einem PL/1-Compiler verarbeitet werden können.

Darüberhinaus überprüft TSTSCAN Modellprogramme auf die Einhaltung der PLIM-Syntax. Im einzelnen wird geprüft, ob

- Bewertungsstatements formal richtig sind,
- die benutzten PLIM-Funktionen im Katalog der verfügbaren PLIM-Funktionen enthalten sind,
- der statische Programmaufbau den Vorschriften entspricht,
- nicht mehrere Bewertungsstatements in einer Programmzeile stehen,
- sich keine Bewertungsstatements innerhalb von ON-Statements, Blöcken oder Unterprogrammen befinden,
- unerlaubte $-Zeichen im Quellcode vorhanden sind.

Dem Preprocessor TSTSCAN kann ein Parameter-String folgenden Inhalts übergeben werden:

- SORMGIN = (xxx-yyy)
  001 < xxx < 009 kennzeichnet die erste Spalte,
  072 < yyy < 080 kennzeichnet die letzte Spalte,

  innerhalb der der Quellcode abgearbeitet wird.
  (Defaultwert: SORMGIN=(002-072))

- TEST

  Das von TSTSCAN reduzierte Modellprogramm wird über 'SYSPRINT' ausgegeben.

- SYSIN HELP FFF NOT

  Für die Programmwartung vorgesehene Parameter.

Bei Aufruf gibt TSTSCAN die folgende Überschrift aus:

'TSTSCAN-MELDUNGEN'

Im Verlauf der Prüfung einer Modellprozedur wird bei jeder Beanstandung eine Botschaft ausgegeben. Eine Botschaft besteht aus:

- der Fehlercodenummer (TSTxxx)
- der Fehlerklasse (I,W,E,S oder T)

  I: informatory
  W: warning
  E: error
  S: severe error
  T: terminating error

- dem Botschaftstext
- der betreffenden Zeile der Eingabedatei
- dem Zeiger 'X', der das beanstandete Zeichen in der betreffenden Zeile markiert.

Beispiel:

```
TST311E '$BEW' - BEWERTUNG ENTHÄLT UNZULÄSSIGE SONDERZEICHEN
    $10:          i = 0;
       X
```

Programmbenutzung unter TSO

Der Aufruf des TSTSCAN erfolgt durch die Kommandoprozedur TSTCOMP, die in der zentralen Datei:
'TSØ045.PRESCAN.CLIST' abgelegt ist (Kap.6).

Die Prozedur besitzt als:

Positionsparameter

- p: Name der Datei, in der die zu testende Modellprozedur abgelegt ist.

Keywordparameter

- MACRO(op)
  op=M falls im Modellprogramm PL/1-Preprocessor-Statements enthalten sind (kein Defaultwert).

- Output(op)
  op gibt den Dateinamen des von TSTSCAN erzeugten Zwischencodes an.
  (Defaultwert: TST)

- DELETE(op)
  op=D; die durch Output bezeichnete Datei wird nach der Compilierung gelöscht.
  op=E; die OUTPUT-Datei wird nicht gelöscht
  (Defaultwert: D)

- STRING(op)
  op enthält eine Zeichenkette (max. 1oo Zeichen) zur Parameterübergabe an TSTSCAN.
  (kein Defaultwert)

Beispiele:

```
exec    TSTCOMP 'MODEL.PLI(MP5) macro(M) -
output  (TEST.PLI) delete(E) string(SORMGIN=(oo9_o72))

exec    TSTCOMP '''TSØXXX.MODEL.PLI(MP2)'' -
string  (SORMGIN=(oo2-o72)-TEST)'
```

Wirkungsweise:

Die Kommandoprozedur ruft nach dem TSTSCAN-Programm den PLIC-Compiler für das von TSTSCAN erzeugte Quellprogramm auf, falls TSTSCAN keine Fehler der Klasse 'severe' bzw. 'terminating' gefunden hat.

Bei fehlerhafter Beendigung von TSTCOMP muß die Kommandoprozedur 'TSØ045.PRESCAN.CLIST(TSTNORM)' aktiviert werden, die alle bestehenden Allokationen aufhebt und temporär angelegte Dateien löscht.

5.4.2. PRESCAN - ein Preprocessor für die Generierung von Inline-Modellprogrammen

Der PLIM-Preprocessor PRESCAN wandelt ein Modellprogramm, das in der PLIM-Maschinensprache (s. Kap.4) geschrieben ist, in ein compilierbares PL/1-Programm um. Programme, die unter Einschaltung des PLIM-Preprocessors PRESCAN übersetzt wurden, können nur noch inline, d.h. unter Kontrolle der PL/1-Maschine, ausgeführt werden. Zur Fehlersuche kann der vom PLIM-Preprocessor erzeugte Zwischencode herangezogen werden. Das Starten des PLIM-Preprocessors sowie das schrittweise Austesten von Modellprogrammen wird in Kap. 7.1. beschrieben. Der PLIM-Preprocessor kennt zwei unterschiedliche Betriebsarten. Im allgemeinen generiert er einen Zwischencode für die Compilierung von Modellprogrammen. Durch Angabe der Option 'MAIN' bzw. 'EXTERNAL' im Prozedurkopf können auch die MAIN-Prozedur bzw. Extern-Prozeduren, die keine eigentlichen Modellprogramme sind, aber Bibliotheksfunktionen oder PLIM-Funktionen benutzen, übersetzt werden.

Aufbereitung von Modellprogrammen

Der PLIM-Preprocessor PRESCAN hat in dieser Betriebsart die Aufgabe, Modellprogramme so zu verändern, daß die darin enthaltenen PLIM-Instruktionen statementweise, zeitlich kontrolliert ausgeführt werden können. Desweiteren muß die Möglichkeit gegeben sein, Modellprogramme reentrant zu benutzen. Der

vom Anwender gewollte Programmablauf darf durch diese Programmerweiterungen nicht verfälscht werden. Ein modifiziertes Modellprogramm, das den obigen Forderungen genügt, besitzt die in Abb. 5.4.-1 dargestellte Struktur. Die vom Benutzer geschriebenen Instruktionen sind darin mit einem Stern gekennzeichnet. Im folgenden werden die Modifikationen und die damit definierten Schnittstellen zur PL/1-Maschine beschrieben.

Modifizierter Prozedurkopf:

Die Options 'TEST', 'NUM' oder 'EXTERNAL' werden aus dem Prozedurkopf gestrichen. Der modifizierte Prozedurkopf von Modellprogrammen erhält folgende Form:

```
MP_n: PROC ($P) RECURSIVE;
```

- n gibt die Nummer des Modellprogramms an.
- Der Pointer $P enthält bei Aufruf die Adresse der DSA des aufrufenden Modellprogramms. Falls keine existiert wird der NULL-Pointer übergeben. Dieser Pointer wird zur Zeit der Programminitialisierung durch die Adresse der aktuellen DSA ersetzt.
- Die Option RECURSIVE ist aus OS/360 betriebsinternen Gründen erforderlich.

Deklaration der statischen Save Area (STSA):

Die STSA ist modellprogrammspezifisch und dient zur Initialisierung der DSA. Es folgt die Struktur der STSA:

```
DCL  1  $STSA          STATIC,
     2  $LAB           LABEL,
     2  $PHSA          POINTER,
     2  $PARM          POINTER,
     2  $NAME          BIN FIXED INIT (n),
     2  $BEW           BIN FIXED INIT (0),
     2  $CPUZEIT       BIN FIXED(31) INIT(0),
     2  $INDEX         BIN FIXED(31) INIT(0),
     2  $ST            BIN FIXED(31) INIT(0),
     2  $PS_START      BIN FIXED(31) INIT(0),
```

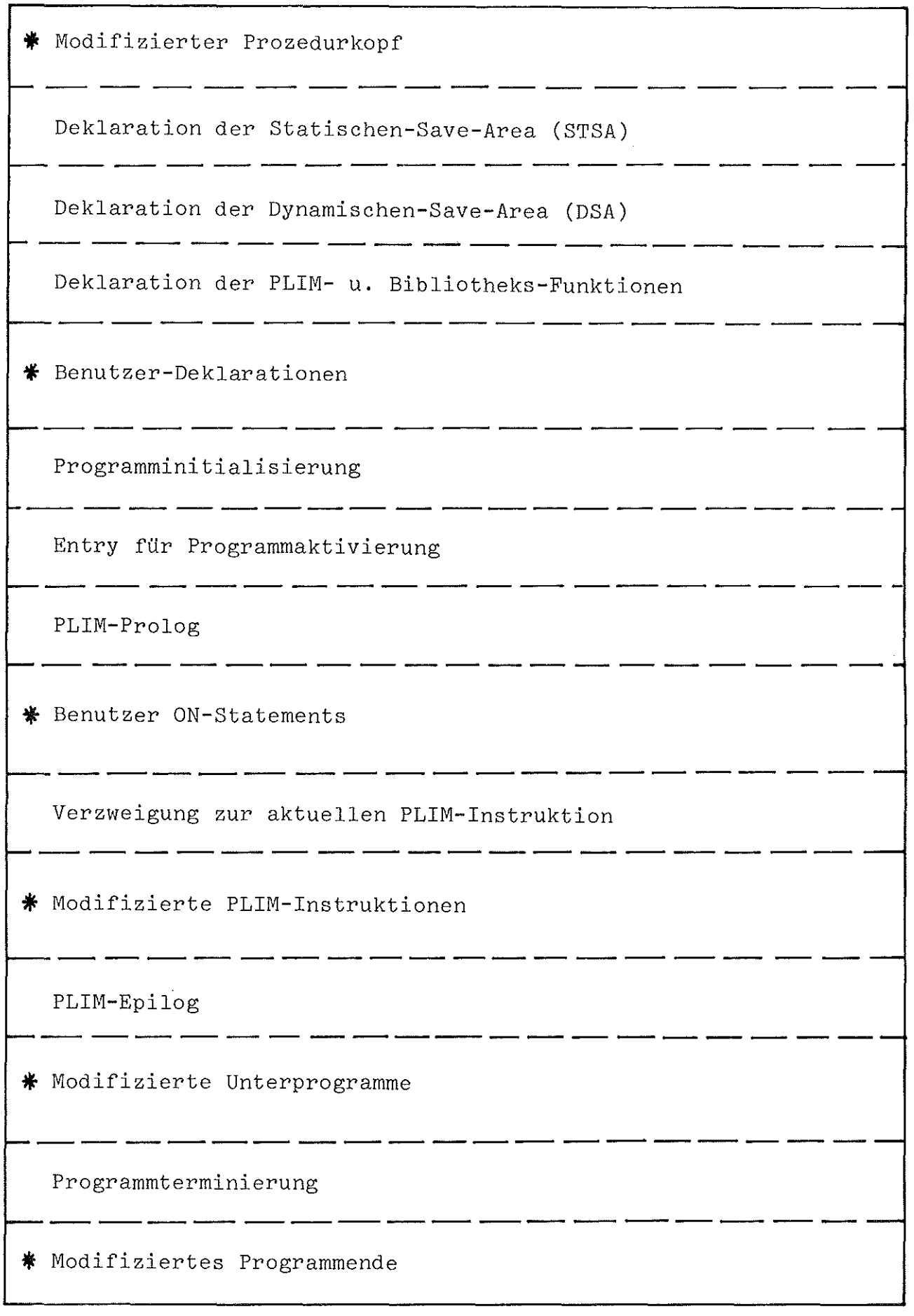


Abb.: 5.4.-1 Modifiziertes Modellprogramm

```
      2  $PS_REST        BIN FIXED(31) INIT(O),
      2  $PROG_ST,
      3  $FINE           BIT(1) INIT('O'B),
      3  $PSEUDO         BIT(1) INIT('O'B),
      3  $TEST           BIT(1) INIT('O'B oder '1'B),
      2  $PF(1o)         POINTER;
```

(n: Nummer des Modellprogramms)

Die Bedeutung der einzelnen Strukturelemente wird gemeinsam mit den Elementen der DSA erläutert.

Deklaration der dynamischen Save Area (DSA)

In der DSA wird der aktuelle Programmstatus abgespeichert. Sie bietet außerdem die Voraussetzung, Modellprogramme reentrant auszuführen. Die Allokation der DSA geschieht zur Zeit der Programminitialisierung. Die Strukturelemente und ihre Bedeutung sind:

```
DCL   1  $DSA         BASED($P),
      2  $LAB         LABEL,
      2  $PHSA        POINTER,
      2  $PARM        POINTER,
      2  $NAME        BIN FIXED,
      2  $BEW         BIN FIXED,
      2  $CPUZEIT     BIN FIXED(31),
      2  $INDEX       BIN FIXED(31),
      2  $ST          BIN FIXED(31),
      2  $PS_START    BIN FIXED(31),
      2  $PS_REST     BIN FIXED(31),
      2  $PROG_ST,
      3  $FINE        BIT(1),
      3  $PSEUDO      BIT(1),
      3  $TEST        BIT(1),
      2  $PF(1o)      POINTER;
```

- $LAB: Labelvariable zur Speicherung der Adresse der folgenden PLIM-Instruktion.
- $PHSA: Pointer zur nächst höheren DSA einer Aufruffolge.
- $PARM: Pointer zu einem Speicherbereich, der zur Parameterübergabe dient. Er wird mit Hilfe der Maschineninstruktion $$STOR_P und $$LOAD_P eingetragen bzw. übergeben.

-$NAME: Nummer des Modellprogramms.

-$BEW: Bewertungsvariable; sie dient zur Speicherung der Ausführungsdauer derjenigen PLIM-Instruktion, auf die $LAB zeigt.

-$CPUZEIT: Aufgelaufene Rechenzeit der Programmfolge.

-$INDEX: Aufrufindex eines Modellprogramms. Die Anzahl der Initialisierungen wird in $STSA.$INDEX aufsummiert.

-$ST: Nummer der PLIM-Instruktion, auf die $LAB zeigt. Sie wird vom PLIM-Preprocessor vergeben und dient zur Programmverfolgung im TEST-Modus.

-$PS_START: Startzeit einer Pseudoaktivphase.

-$PS_REST: Restliche Pseudoaktivzeit bei Programmunterbrechungen.

-$PROG_ST: Programmstatus.

-$FINE: Kennzeichen für die Beendigung eines Modellprogramms. Es wird zur Zeit der Programmterminierung zu eins gesetzt.

-$PSEUDO: Kennzeichen für den Pseudoaktivstatus eines Modellprogramms. Es wird durch die $$RUN-Funktion gesetzt.

-$TEST: Kennzeichen für den TEST-Status. Es wird mit $$TEST(B) gesetzt und mit $$TEST(E) gelöscht.

-$PF(lo): Feld zum Speichern von BASED-Pointern.Sie werden im PLIM-Prolog zugewiesen und im PLIM-Epilog in die DSA gerettet.

Sowohl die STSA als auch die DSA werden erst zur Compile-Zeit in die Modellprogramme durch %INCLUDE... eingefügt. Das bedeutet, daß die Schnittstelle zwischen PL/1-Maschine und Modell variiert werden kann, ohne den PLIM-Preprocessor zu verändern.

Deklaration der PLIM- und Bibliotheksfunktionen:

Die Deklarationen der PLIM- und Bibliotheksfunktionen wird wie die STSA und DSA zur Compile-Zeit in Modellprogramme eingefügt.

Benutzer-Deklarationen:

Die Benutzer-Deklarationen werden nach BASED-Pointern durchsucht. Die dabei ermittelten ersten zehn Pointer werden im PLIM-Prolog bzw. Epilog automatisch verwaltet.

Programminitialisierung:

Bevor ein Modellprogramm, d.h. die erste PLIM-Instruktion eines Modellprogramms, aktiviert werden kann, muß es durch Aufruf des Prozedurnamens initialisiert werden. Dabei wird zunächst die DSA angelegt und der Aufrufindex in der STSA erhöht. Der Pointer $PHSA erhält den mit dem Parameter $P übergebenen Wert, $LAB wird auf die erste PLIM-Instruktion eingestellt. Die restlichen Elemente der DSA werden aus der STSA kopiert. Der Pointer $P erhält die Adresse der aktuellen DSA.

Entry für die Programmaktivierung:

Zur Aktivierung der PLIM-Instruktionen wird ein zweiter Entry in Modellprogrammen eingefügt. Er besitzt die Form:

```
MP_nE:  ENTRY ($P);
```

Der Pointer $P zeigt auf die aktuelle DSA.

PLIM-Prolog:

Im PLIM-Prolog werden max. 10 vom Benutzer deklarierte BASED-Pointer und die Label-Variable $DSA.$LAB aktualisiert.

Benutzer ON-Statements:

Die vorgeschriebene Position der ON-Statements gewährleistet ihre Gültigkeit für alle PLIM-Instruktionen eines Modellprogramms. Die in den ON-Statements enthaltenen Ausführungsstatements werden vom Preprocessor modifiziert.

Verzweigung zur aktuellen PLIM-Instruktion:

An dieser Stelle eines Modellprogramms wird zum Ausführungsstatement der aktuellen PLIM-Instruktion verzweigt.

Modifizierte PLIM-Instruktion:

Sie besitzt die Form:

```
[Benutzer-Label:]...
$DSA.$BEW = Bewertung;
$DSA.$LAB = $Lm;
$DSA.$ST  = m;
 GOTO  PLIM-Epilog;
$Lm:
[Modifiziertes Ausführungsstatement]
```

- m bedeutet hier die Nummer der PLIM-Instruktion. Sie wird vom PLIM-Preprocessor vergeben.
- Die $$-Zeichen der PLIM- und Bibliotheksfunktionen im Ausführungsstatement werden durch ein $-Zeichen ersetzt.
- Vor den Maschineninstruktionen I, Pseudoinstruktionen I und Bibliotheksfunktionen I wird 'CALL' eingefügt.
- Falls im Ausführungsstatement ein 'RETURN'; enthalten ist, wird zur Programmterminierung verzweigt.

PLIM-Epilog:

Im PLIM-Epilog werden die BASED-Pointer des Modellprogramms in die DSA gerettet. Das Modellprogramm wird via RETURN verlassen.

Modifizierte Unterprogramme:

In Unterprogrammen werden Ausführungsstatements entsprechend den PLIM-Konventionen modifiziert.

Programmterminierung:

In der Programmterminierung wird das $FIN-Bit in der DSA gesetzt. Die Freigabe der DSA wird von der PL/1-Maschine durchgeführt.

Modifiziertes Programmende:

Vor dem END-Statement eines Modellprogramms wird zur Programmterminierung verzweigt.

Verarbeitung von Programmen mit der Option MAIN bzw. EXTERNAL:

Die Prozedur mit der Option MAIN darf Bibliotheksfunktionen aufrufen. In Prozeduren mit der Option EXTERNAL können alle PLIM- und Bibliotheksfunktionen benutzt werden. Die hierfür nötigen Deklarationen und die Modifikation der Ausführungsstatements werden vom PLIM-Preprocessor durchgeführt. Bewertungsstatements sind in dieser Klasse von Programmen nicht erlaubt.

6. Dateiorganisation unter TSO

Die zum Simulationssystem gehörigen Programme, Daten und Kommandoprozeduren sind entsprechend ihrer Funktion in untergliederten Dateien, den sog. Partitioned Data Sets (PDS), zusammengefaßt. Ein PDS besteht aus einem Inhaltsverzeichnis, sowie aus einzelnen, sequentiell aufgebauten Gliedern den sog. Members. Die PDS-Namen der Programme des Simulationssystems setzen sich zusammen aus den TSO-Benutzernamen, den Funktionsnamen und dem Qualifikationsbegriff. Alle Programme, Daten und Kommandoprozeduren des Simulationssystems werden

in einer TSO-Datei verwaltet und besitzen deshalb den gleichen TSO-Benutzernamen (TSØ045). Die folgenden allgemeinen Dateinamen sind z.Zt. vergeben:

- EVSIM — für den Bereich der Eventsteuerung und der Bibliotheksfunktionen
- PLIMSIM — für die PL/1-Maschine
- PRESCAN — für die Preprocessoren und die Modellprogrammerstellung

z.B.:

a) TSØ045.PRESCAN.LOAD(TSTSCAN): Load-Modul des TSTSCAN
b) TSØ045.PRESCAN.CLIST(TSTCOMP): Kommandoprozedur zum Starten des TSTSCAN
c) TSØXXX.MODELL.PLI(MP1): Modellprogramm MP_1 in dem PDS TSØXXX.MODELL.PLI

EVSIM-Dateien

Unter dem Funktionsbegriff EVSIM werden alle Dateien geführt, die zur Eventsteuerung gehören. Sie sind in / 1 u. 2 / beschrieben.

PLIMSIM-Dateien

Die zur PLIM gehörigen Dateien sind mit dem Funktionsbegriff PLIMSIM gekennzeichnet (s. Kap. 5.3.).

PRESCAN-Dateien

Zu den PRESCAN-Dateien gehören die Preprocessoren TSTSCAN und PRESCAN, sowie Kommandoprozeduren und Hilfsprogramme für deren Benutzung / 4 /. Auf die Handhabung des TSTSCAN für den Offline Test wurde in Kap. 5.4.1. eingegangen. Die Bedienung des Preprocessors PRESCAN wird in Kap. 7.1. beschrieben.

Modellprogramm-Dateien

Der Quellcode von Modellprogrammen kann vom Benutzer unter beliebigen Data Set Namen erstellt werden. Vorgeschrieben sind jedoch die Record-Länge (LRECL=80) und das Record-Format (RECFM=FB). Die Source Margin sollte i.a. 'SORMGIN' = (002-072) und die Datei sollte zeilennumeriert sein. Vorgeschrieben ist ebenfalls die Organisation der Objekt-Datei aller Modellprogramme, die zu einem Simulationslauf gehören. Sie sind Member eines PDS mit der Qualifikation OBJ. Die Membernamen von Modellprogrammen müssen aus den Prozedurnamen wie folgt abgeleitet werden. Modellprogramme mit den Prozedurnamen MP_n (n = 1,2 .... 99) erhalten die Membernamen MPn. Zum Beispiel:

```
MP_1: PROC; ⟶  TSOxxx.yyy.OBJ(MP1)
MP_2: PROC; ⟶  TSOxxx.yyy.OBJ(MP2)
```

usw.

- TSOxxx ist die TSO-Benutzeridentifikation des jeweiligen Benutzers
- yyy ist ein frei wählbarer Funktionsname
- Der Membername der MAIN-Prozedur oder von EXTERNAL-Prozeduren ist beliebig.

7. Bedienung der PLIM unter TSO

Ein Modell für die PLIM besteht aus einem Hauptprogramm (Option MAIN) und einer Reihe von Modellprogrammen.

In diesem Kapitel werden Form und Aufbau dieser Programme beschrieben sowie Hinweise für die stufenweise Vorbereitung eines Simulationslaufs gegeben. Die Einbettung des Modells in das Gesamtsystem kann dazu noch einmal anhand der Abb. 7-1 überprüft werden, die die globale Verkettung aller externen Prozeduren zeigt.

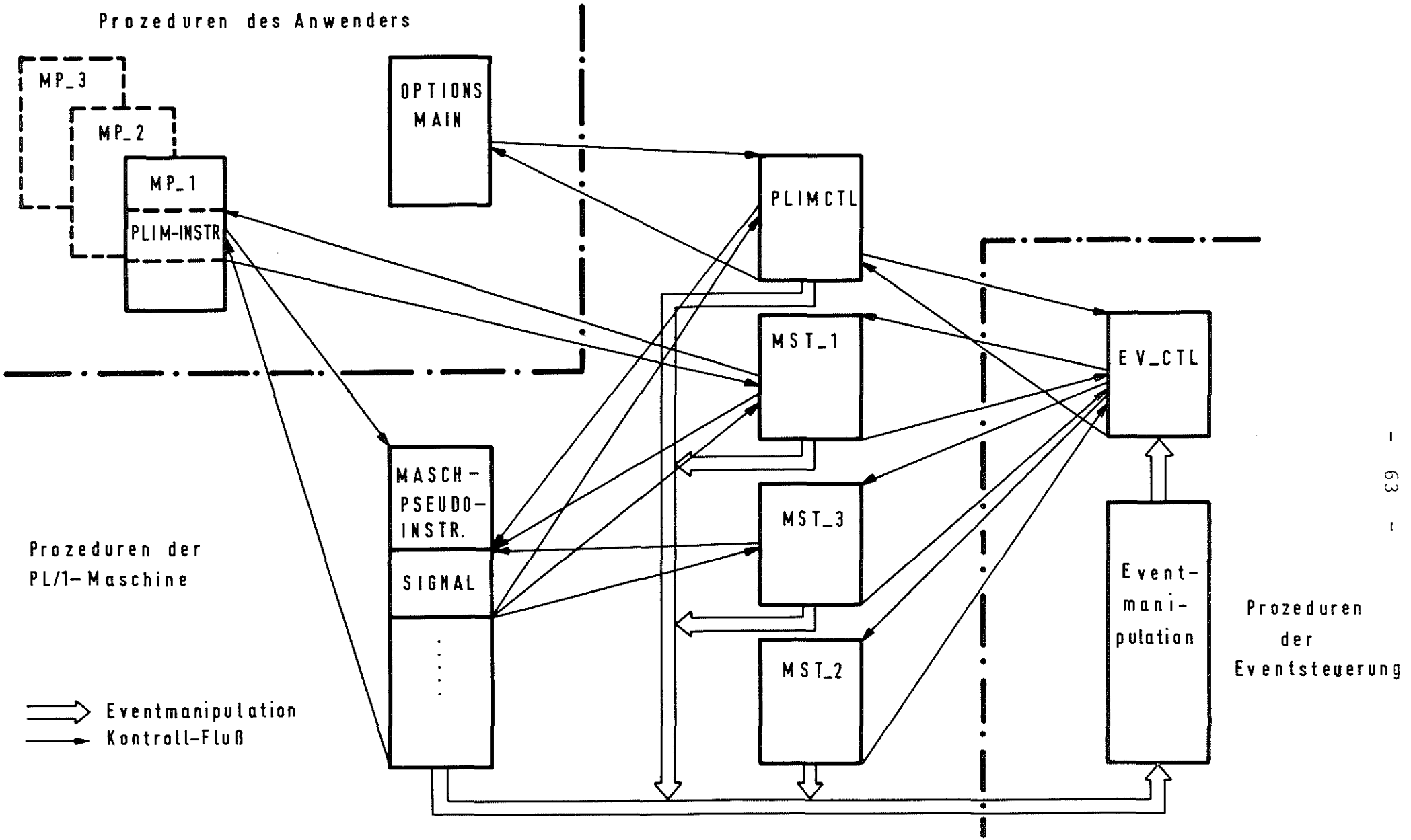


Abb.: 7-1 Funktionelle Abhängigkeit der Prozeduren im Simulationssystem

7.1. Modellaufbau, -test und -präparierung

Die nachstehend aufgeführte PL/1-Programmfolge zeigt eine typische MAIN-Prozedur für die PLIM:

```
DISPATCHER: PROCEDURE OPTIONS(MAIN);
DCL  PLIMCTL  ENTRY;
   |
   |restlicher
   |Deklarationsteil
   |
   |
   |Modelldefinition u.
   |-initialisierung
   |
CALL PLIMCTL;
   |
   |Auswertung
   |
END DISPATCHER;
```

Im Deklarationsteil der MAIN-Prozedur, deren Name frei wählbar ist, muß die externe Prozedur PLIMCTL aufgeführt sein, die die Verbindung des benutzerspezifischen Modellprogrammsystems mit der PLIM herstellt. Die MAIN-Prozedur zerfällt grob in drei Teile.

Zuerst erfolgt die benutzerspezifische Definition und Initialisierung des Modells.

Durch den Aufruf von PLIMCTL wird die Konfigurations- und -startphase der PLIM eingeleitet, in der die Zahl der Rechnerkerne kreiert sowie Priorität und Startadresse der Interruptprogramme geladen werden.

Nach Rückgabe der Kontrolle an die MAIN-Prozedur, die entweder durch gezielten Abbruch oder fehlerhaften Lauf im Testfall ausgelöst werden kann, wird in der Regel eine Auswertungsphase angeschlossen.

Innerhalb der MAIN-Prozedur dürfen lediglich die Zufallszahlengeneratoren (Bibliotheksfunktionen) benutzt werden, nicht aber die PLIM-Funktionen (vgl. hierzu die Syntaxvorschriften in Kap. 4).

Alle Modellprogramme haben folgendes Aussehen:

```
MP_x: PROCEDURE;
.
.
.
END MP_x;
```

Der interne Aufbau sowie die Syntaxvorschriften für den Gebrauch der PLIM-Instruktionen, der hier in vollem Umfange möglich ist, sind ausführlich unter 4. beschrieben. Bevor die Integration der Modellprogramme in die PLIM vorgenommen wird, sollte sichergestellt werden, daß sie syntaktisch einwandfrei sind. Außerdem sollte darauf geachtet werden, daß alle Maschineninstruktionen mit Zeitbewertungen versehen sind, um eine schritthaltende Synchronisation zu gewährleisten. Das Zusammenfassen mehrerer Maschineninstruktionen in einer PLIM-Instruktion kann zu logischen Fehlern im Programmablauf führen.

Für den Inline-Test sollten alle Modellprogramme vom PLIM-Preprocessor unter Angabe der TEST-Option übersetzt werden. Dadurch werden alle PLIM-Instruktionen durchnumeriert und eine ON-Unit der Form

```
ON ERROR    $$STOP;
```

in den modifizierten Quellcode eingefügt.

Durch Ausführung der Pseudoinstruktion $$TEST während der Simulation wird eine Testphase eingeleitet, in der der Ablauf aller Modellprogramme instruktionsweise aufgelistet wird.

Daneben besteht die Möglichkeit, beim Eintritt bestimmter Situationen durch Ausführung der $$RKDUMP-Instruktion den aktuellen Inhalt aller Rechnerkerne auszugeben.

Der Offline-Test wird mit dem TSTSCAN-Preprocessor durchgeführt, dessen Handhabung in Kap. 5.4.1. ausführlich beschrieben ist. Dabei sollte vor allem die Cross Reference-Table mit den im Modellprogramm deklarierten Variablen verglichen werden. Die Verwendung nicht explizit deklarierter Größen führt meist zu schwer auffindbaren Fehlern.

Für die Generierung des Inline-Codes von Modellprogrammen wird die Kommandoprozedur PRCOMP aufgerufen. Sie ist unter dem Data Set Namen

'TSØ045.PRESCAN.CLIST(PRCOMP)'

abgelegt und sollte vor Benutzung in die benutzereigene Datei kopiert werden. Durch den Start der Kommandoprozedur wird zunächst der PLIM-Preprocessor und darauffolgend der PL/1-Optimizing Compiler aufgerufen. Die Prozedur besitzt als:

Positionsparameter

- p: Name der Eingabedatei mit dem Quellcode eines Modellprogramms. Der DS-Name muß voll qualifiziert angegeben werden.

  z.B.: TSOxxx.MODELL.PLI(MP1)

Die folgenden Keywordparameter müssen angegeben werden:

- OBJ(op)

  op gibt den PDS-Namen der OBJ-Datei an, in die der Objekt-Code abgelegt werden soll.
  (Defaultwert: xxx)

- MEM(op)

  op gibt den Membernamen in der OBJ-Datei an, unter welcher der vom PL/1-Optimizing-Compiler erzeugte Objekt-Code abgelegt wird.
  (Defaultwert: MP1)

Die folgenden Keywordparameter können bei Bedarf überschrieben werden:

- OUT(op)
  op gibt den Namen eines sequentiellen Data Set an, unter dem der Zwischencode abgelegt wird.
  (Defaultwert: OUT.TEMP)

- SORMGIN(op)
  op beschreibt die Source Margin des Quellcodes wie folgt:

  ```
  op = 'SORMGIN = (Ka-Ke) mit
       001 < Ka < 009
       072 < Ke < 080
  ```

  (Defaultwert: SORMGIN = (002-072))

- NUM(op)
  op = NUM der Zwischencode wird in den Spalten 72-80 zeilennumeriert
  op = NO   der Zwischencode wird nicht zeilennumeriert
  (Defaultwert: NO)

- TEST(op)
  op = TEST das Modellprogramm ist im Testmodus ausführbar
  op = NO   das Modellprogramm kann im Testmodus nicht ausgeführt werden
  (Defaultwert: TEST)

- D(op)
  op = D   der Zwischencode wird gelöscht
  op = E   der Zwischencode wird nicht gelöscht
  (Defaultwert: D)

- F1(op)
  op gibt den Namen der Include-Datei mit dem File-Namen PRESCAN an
  (Defaultwert: TSØ045.PRESCAN.PLI)

- F2(op)
  op gibt den Namen der Include-Datei mit dem File-Namen EVSIM an.

  (Defaultwert: TSØ045.EVSIM.PLI)

- F3(op)
  op gibt den Namen der Include-Datei mit dem File-Namen PLIMSIM an.

  (Defaultwert: TSØ045.PLIMSIM.PLI)

Bei fehlerhafter Beendigung der Prozedur PRCOMP oder bei D(E) muß die Kommandoprozedur PRNORM gestartet werden, bevor PRCOMP erneut aufgerufen werden kann.

Beispiele:

```
exec  'TSØ045.PRESCAN.CLIST(PRCOMP)' 'TSØXXX.MODELL.PLI(MP1)-
       test(TEST) obj(MODELLA) mem(MP1)-
       d(E) num(NO)'

exec  'TSØ045.PRESCAN.CLIST(PRNORM)'

exec  'TSØ045.PRESCAN.CLIST(PRCOMP)' 'TSØXXX.MOD.PLI(MP15)-
       obj(MOD) mem(MP15) test(NO) num(NO)'

exec  PRCOMP 'TSØXXX.MOD.PLI(MP15) obj(MOD) mem(MP15)'
      (wenn PRCOMP in die Benutzer-Datei kopiert wurde)
```

7.2. Systemgenerierung, Link und Start

Zur Systemgenerierung, Link und Start, des Simulationssystems stehen folgende Kommandoprozeduren zur Verfügung:

PLIMC
PLIMCL
PLIML
PLIMCLG
PLIMLG
PLIMNORM

Die Prozeduren sind Member des PDS:

TSØ045.PLIMSIM.CLIST

Um eine parallele Benutzung der Kommandoprozeduren zu ermöglichen, müssen die Prozeduren vor Gebrauch in eine anwendereigene TSO-Datei kopiert werden.

Die Kommandoprozedur PLIMC

Die Prozedur wird zur Vorbereitung des Link-Steps (Kommandoprozedur PLIML/PLIMLG) benötigt. Sie besitzt folgenden Keyword-Parameter:

- MPANZ(op)
  op gibt die Anzahl der Modellprogramme an, die in das Gesamtsystem integriert werden sollen.

  Dabei wird vorausgesetzt:

  1) die Modellprogramme dürfen nur Namen der Form MP_i (i=1..MPANZ) tragen;

  2) die Modellprogramme müssen übersetzt als Member eines PDS vorliegen. Als Membernamen sind nur Namen der Form MPi (i=1..MPANZ) erlaubt.

  (Defaultwert: 1)

  Beispiel:

  ```
  exec  PLIMC  'mpanz(13)'
  ```

  Wirkungsweise:

  Die Kommandoprozedur erzeugt folgende drei Datensätze in der TSO-Datei des Anwenders:

  PLIMSIM.C1.DATA
  PLIMSIM.C2.OBJ
  PLIMSIM.C3.OBJ

Die Existenz dieser Datensätze wird bei Benutzung der Kommandoprozeduren PLIML und PLIMLG vorausgesetzt. Sie dienen dem Linkage Editor als 'primary-' bzw. als 'secondary Input'.

Inhalt und Bedeutung der Datensätze:

- PLIMSIM.C1.DATA
  enthält INCLUDE-Statements, die die Membernamen der übersetzten Modellprogramme referieren.
- PLIMSIM.C2.OBJ
  enthält den Objektcode der PLIM-Hilfsfunktion CALL_MP. Die Prozedur fungiert als Schnittstelle zwischen PLIM-Prozeduren und Modellprogrammen. Anwendungsabhängig wurden in den Quellcode der Prozedur mit Hilfe des PL/1-Makroprozessors die Deklarationen der Modellprogramme und ein Sprungverteiler zum Aufruf der Modellprogramme eingefügt.
- PLIMSIM.C3.OBJ
  enthält den Objektcode der Eventkontrolle, die nach Aufbereitung durch den PL/1-Makroprozessor als Schnittstelle zwischen Eventsteuerung und Eventprozeduren dient.

Die Kommandoprozedur PLIMCL

Die Prozedur dient zum Erstellen eines modellspezifischen Loadmoduls.

Der Loadmodul ist vom Linkage-Editor wiederverarbeitbar, so daß noch benutzereigene externe Prozeduren eingebunden werden können.

Folgende Keyword-Parameter müssen angegeben werden:

- MPANZ(op)
  s. Kommandoprozedur PLIMC
  (Defaultwert: 1)
- MPOBJ(op)
  op gibt den Namen des PDS an, dessen Member die Objektmodule der Modellprogramme enthalten.
  (kein Defaultwert)

- MAINOBJ(op)
  op gibt den Namen des Datensatzes an, der den Objektcode des Hauptprogramms enthält.
  (kein Defaultwert)

- LOADM(op)
  op gibt den Namen der Load-Datei an, die den erzeugten Loadmodul aufnimmt.
  (Defaultwert: SIM(TEMPNAME))

Beispiel:

```
exec  PLIMCL 'mpobj(MP.OBJ) mpanz(8) mainobj(HAUPT.OBJ)-
             loadm(PLIM)'

ex    PLIMCL 'mpanz(8) mpobj("TSØxxx.yy.OBJ")-
             mainobj(MAIN.OBJ)'
```

Wirkungsweise:

Das Ablaufschema der Prozedur PLIMCL wird in Abb. 7-2 dargestellt.

Die Prozedur ist in zwei Teile gegliedert:

a) Vorbereitung des Linksteps
b) Ausführung des Linksteps

zu a)

Die Ausführungsfolge ist identisch mit der der Kommandoprozedur PLIMC. Die erzeugten Dateien werden temporär angelegt, d.h. sie werden nach Ausführung des Linksteps gelöscht.

zu b)

Zusammen mit den erzeugten Dateien werden dem Linkage-Editor übergeben:

1) der Objektmodul des Hauptprogramms
2) als 'secondary-Input' ein Datensatz mit INCLUDE-Statements, die die Verbindung zu den Objektmoduln der

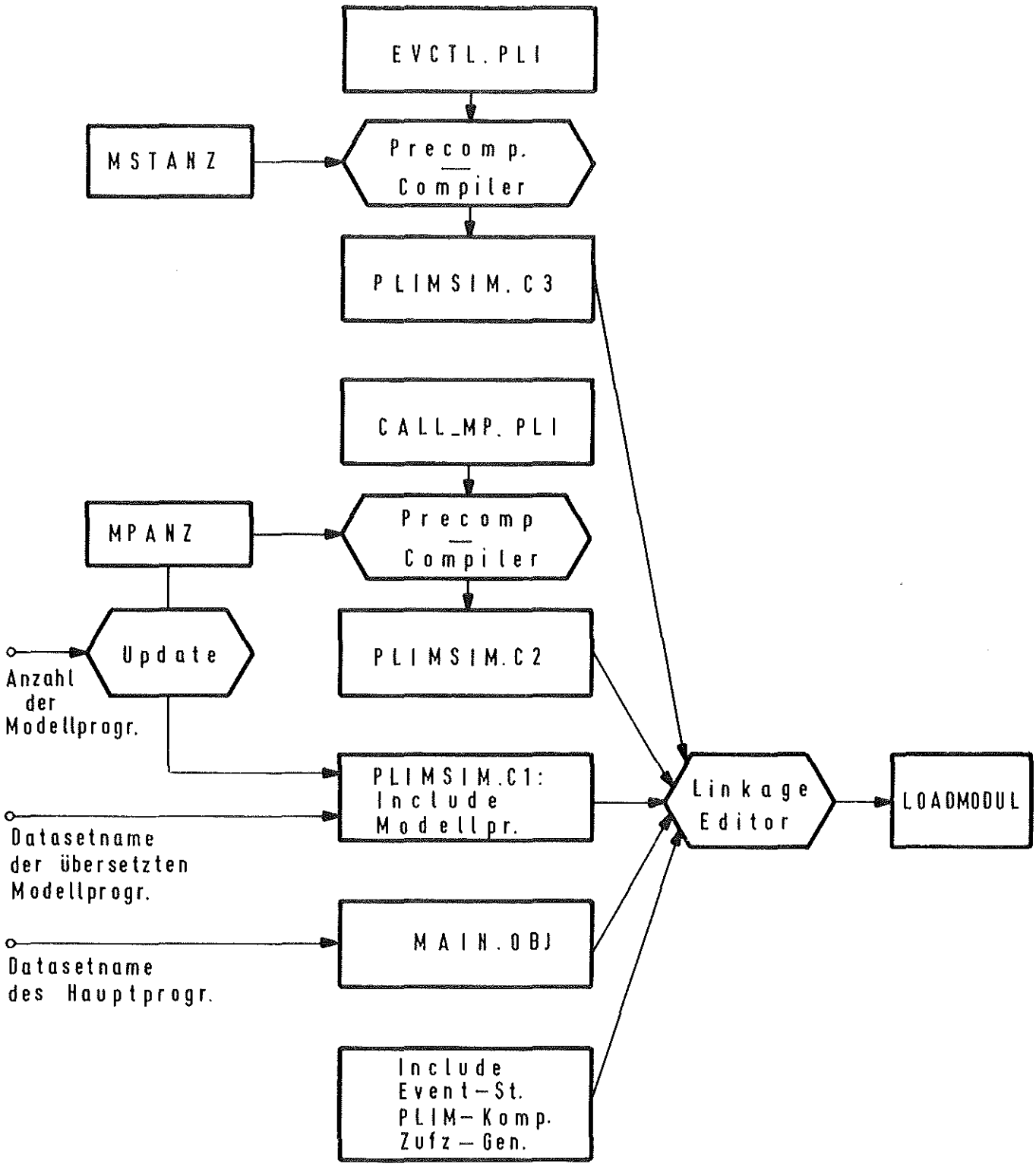


Abb.: 7-2 Ablaufschema der Kommandoprozedur PLIM CL

- Eventsteuerung
- PLIM-Prozeduren
- Zufallszahlengeneratoren

herstellen.

Der Linkage-Editor bindet die auf diese Weise direkt oder indirekt referierten Objektmodule zu einem Loadmodul zusammen. Falls keine benutzereigene externe Prozeduren anzufügen sind, ist der Loadmodul sofort ausführbar.

Die Kommandoprozedur PLIML

Die Kommandoprozedur PLIML erstellt einen wiederverarbeitbaren Loadmodul des Simulationssystems.

Die Prozedur setzt die Existenz der von der Kommandoprozedur PLIMC erzeugten Datensätze voraus.

PLIML hat zusammen mit PLIMC die gleiche Wirkung wie die Kommandoprozedur PLIMCL.

Die Prozedur besitzt folgende Keyword-Parameter:

- MPOBJ
- MAINOBJ
- LOADM

Die Parameter sind gemäß den Vorschriften des vorigen Abschnitts anzuwenden.

Beispiel:

```
exec  PLIML 'mpobj(MP.OBJ) mainobj(SIM.OBJ)-
            loadm(LOAD1)'

exec  PLIML 'mpobj(MODEL.OBJ) mainobj(MODEL.OBJ(MAIN))-
            loadm(L(M1))'
```

Die Kommandoprozedur PLIMCLG

Die Prozedur PLIMCLG ist eine Erweiterung der Prozedur PLIMCL. Mit Hilfe von PLIMCLG wird ein wiederverarbeitbarer Loadmodul des Simulationssystems erzeugt und anschließend ausgeführt. Folgende Keyword-Parameter werden benötigt:

- MPANZ
- MPOBJ
- MAINOBJ
- LOADM

Diese Parameter sind bedeutungsgleich mit denen der Kommandoprozedur PLIMCL. Zusätzlich können angegeben werden:

- IN(op)
  op bezeichnet den Datensatz, der der System-File "Sysin" zugeordnet wird.
  (Defaultwert: (Terminal))

- OUT(op)
  op bezeichnet den Datensatz, der der System-File "Sysprint" zugeordnet wird.
  (Defaultwert: (Terminal))

Beispiel:

```
ex    PLIMCLG 'mainobj("TSØxxx.yy.OBJ") loadm(LD)-
              mpanz(11) mpobj("TSØxxx.zz.OBJ")'

exec  PLIMCLG 'mpanz(5) mainobj(MAIN) loadm(SIM)-
              in(EING.DATA) out(AUSG.DATA)'
```

Die Kommandoprozedur PLIMLG

Die Prozedur PLIMLG ist eine Erweiterung der Kommandoprozedur PLIML. Sie setzt die Existenz der von der Kommandoprozedur PLIMC erzeugten Datensätze voraus. Die Prozedur erstellt einen wiederverarbeitbaren Loadmodul und führt ihn aus. Es stehen folgende Keyword-Parameter zur Verfügung:

- MPOBJ
- MAINOBJ
- LOADM
- IN
- OUT

Die Parameter sind in den vorigen Abschnitten beschrieben.

Beispiel:

```
exec  PLIMLG 'mpobj(MODEL.OBJ) mainobj(MAIN.OBJ)-
             loadm(LOAD) in(EING.DATA)'

ex    PLIMLG 'mpobj(M1.OBJ) mainobj(M1.OBJ(M))-
             in(A) out(B)'
```

Die Kommandoprozedur PLIMNORM

Wird eine der in diesem Kapitel beschriebenen Kommandoprozeduren vorzeitig abgebrochen, so bleiben im allgemeinen PLIM-spezifische Allokationen und temporär angelegte Dateien bestehen. In diesem Fall werden durch Aufruf der Kommandoprozedur PLIMNORM die bestehenden Allokationen aufgelöst und noch existierende Hilfsdateien gelöscht.

Die Prozedur besitzt keine Parameter.

Beispiel:

```
exec  PLIMNORM
```

8. Beispiel

8.1. Modellerstellung und -lauf

Das vorliegende Beispiel besteht aus dem Hauptprogramm 'MAIN' und den Modellprogrammen MP_1 und MP_2. In den Modellprogrammen werden zwei unterschiedliche Auflösungen einer DO-Gruppe realisiert, während im Hauptprogramm Schleifenanfang und Schleifenende eingelesen werden. Beide Modellprogramme laufen

im TEST-Modus ab.

Liste des Hauptprogramms:

```
list example.pli(main)

 EXAMPLE.PLI(MAIN)
00010  MAIN: PROC OPTIONS(MAIN);
00020  DCL PLIMCTL ENTRY;
00030  DCL (A,B) BIN FIXED EXT;
00040
00050  PUT SKIP LIST('GIB SCHLEIFENANFANG:');
00060  GET LIST(A);
00070  PUT LIST('GIB SCHLEIFENENDE:');
00080  GET LIST(B);
00090  CALL PLIMCTL;
00100
00110  END MAIN;
READY
```

Liste des Modellprogramms MP_1:

```
list example.pli(mp1)

 EXAMPLE.PLI(MP1)
00010  MP_1: PROC;
00020  /*********************************************************************/
00030  /* AUFLOESUNG DER DO-GROUP:    Z=1;                                  */
00040  /*                           DO I=A TO B;                            */
00050  /*                              Z=Z*2;                               */
00060  /*                           END;                                    */
00070  /* MIT NACHGESTELLTER ABFRAGE                                        */
00080  /*********************************************************************/
00090  DCL (A,B) BIN FIXED EXT;
00100  DCL 1 STR BASED(P),
00110        2 Z BIN FIXED(31),
00120        2 I BIN FIXED(15);
00130
00140  $0     ALLOCATE STR;
00150  $0     PUT SKIP(2) LIST('BEGINN: MP_1');
00160         $$TEST('B');
00170
00180  $1     Z=1;
00190  $1     I=A;
00200     M:
00210  $1     Z=Z*2;
00220  $1     I=I+1;
00230  $1     IF I<=B THEN GOTO M;
00240
00250  $0     CALL DRUCK;
00260         $$IDLE;
00270  $0     $$TEST('E');
00280
00290  DRUCK: PROC;
00300  PUT SKIP(2) LIST('ENDE: MP_1');
00310  PUT SKIP(2) EDIT('SCHLEIFENINDEX:',I)(A(15),F(10));
00320  PUT SKIP EDIT('ERGEBNIS:',Z)(A(15),F(10));
00330  PUT SKIP EDIT('CPU-ZEIT:',$$CPUTME)(A(15),F(10));
00340  END DRUCK;
00350
00360  $0     END MP_1;
READY
```

Liste des Modellprogramms MP_2:

```
list example.pli(mp2)

 EXAMPLE.PLI(MP2)
00010  MP_2: PROC;
00020  /*********************************************************************/
00030  /* AUFLOESUNG DER DO-GROUP:    Z=1;                                  */
00040  /*                           DO I=A TO B;                            */
00050  /*                              Z=Z*2;                               */
00060  /*                           END;                                    */
00070  /* MIT VORANGESTELLTER ABFRAGE                                       */
00080  /*********************************************************************/
00090  DCL (A,B) BIN FIXED EXT;
00100  DCL 1 STR BASED(P),
00110        2 Z BIN FIXED(31),
00120        2 I BIN FIXED(15);
00130
00140  $0     ALLOCATE STR;
00150  $0     PUT SKIP(2) LIST('BEGINN: MP_2');
00160         $$TEST('B');
00170
00180  $1     Z=1;
00190  $1     I=A;
00200     M1:
00210  $1     IF I>B THEN GOTO M2;
00220  $1     Z=Z*2;
00230  $1     I=I+1;
00240  $1     GOTO M1;
00250     M2:
00260
00270  $0     CALL DRUCK;
00280  $0     $$TEST('F');
00290  $0     $$STOP;
00300
00310  DRUCK: PROC;
00320  PUT SKIP(2) LIST('ENDE: MP_2');
00330  PUT SKIP(2) EDIT('SCHLEIFENINDEX:',I)(A(15),F(10));
00340  PUT SKIP EDIT('ERGEBNIS:',Z)(A(15),F(10));
00350  PUT SKIP EDIT('CPU-ZEIT:',$$CPUTME)(A(15),F(10));
00360  $$RKDUMP;
00370  END DRUCK;
00380
00390  $0     END MP_2;
READY
```

Die Programme werden übersetzt und ihr Objektcode im PDS 'example.obj' abgelegt.

Übersetzung des Hauptprogramms mittels des TSO-Standard-kommandos:

```
pli example.pli(main) f(s) opt(time) obj(example(main))

OPTIMIZING COMPILER INVOKED
PL/1 OPTIMIZER V1 R1.1 TIME: 11.29.45  DATE: 3 JAN 73
OPTIONS SPECIFIED
 OBJ,F(S),OPT(TIME)



NO MESSAGES OF SEVERITY S AND ABOVE PRODUCED FOR THIS COMPILATION
MESSAGES SUPPRESSED BY THE FLAG OPTION:  1 I.
COMPILE TIME    0.02 MINS          SPILL FILE:      0 RECORDS, SIZE  3491
READY
```

Übersetzung der Modellprogramme mit Hilfe der Kommandoprozedur PRCOMP:

```
exec prcomp 'tso427.example.pli(mp1) obj(example) mem(mp1)'

FILE SYSIN     NOW UNALLOCATED
FILE SYSPRINT NOW UNALLOCATED
FILE OUTPUT    NOW UNALLOCATED
FILE SYSPRINT NOW UNALLOCATED
OPTIMIZING COMPILER INVOKED
PL/1 OPTIMIZER V1 R1.1 TIME: 10.35.28  DATE: 3 JAN 73
OPTIONS SPECIFIED
 M,OBJ,F(S),OPT(TIME)


NO MESSAGES PRODUCED BY THIS MACRO PASS



NO MESSAGES OF SEVERITY S AND ABOVE PRODUCED FOR THIS COMPILATION
MESSAGES SUPPRESSED BY THE FLAG OPTION:  8 W.  4 I.
COMPILE TIME    0.14 MINS          SPILL FILE:      2 RECORDS, SIZE  3491
DATA SET OUT.TEMP.PLI NOW UNALLOCATED
FILE PRESCAN  NOW UNALLOCATED
FILE EVSIM    NOW UNALLOCATED
FILE PLIMSIM  NOW UNALLOCATED
READY
```

```
exec prcomp 'tso427.example.pli(mp2) mem(mp2) obj(example)'

FILE SYSIN    NOW UNALLOCATED
FILE SYSPRINT NOW UNALLOCATED
FILE OUTPUT   NOW UNALLOCATED
FILE SYSPRINT NOW UNALLOCATED
OPTIMIZING COMPILER INVOKED
PL/1 OPTIMIZER V1 R1.1 TIME: 10.37.32  DATE: 3 JAN 73
OPTIONS SPECIFIED
 M,OBJ,F(S),OPT(TIME)


NO MESSAGES PRODUCED BY THIS MACRO PASS




NO MESSAGES OF SEVERITY S AND ABOVE PRODUCED FOR THIS COMPILATION
MESSAGES SUPPRESSED BY THE FLAG OPTION:  8 W.  4 I.
COMPILE TIME    0.15 MINS        SPILL FILE:     4 RECORDS, SIZE  3491
DATA SET OUT.TEMP.PLI NOW UNALLOCATED
FILE PRESCAN  NOW UNALLOCATED
FILE EVSIM    NOW UNALLOCATED
FILE PLIMSIM  NOW UNALLOCATED
READY
```

Zur Vorbereitung des Linksteps wird die Kommandoprozedur PLIMC aufgerufen:

```
exec plimsim(plimc) 'mpanz(2)'


IBM0031I ERRONEOUS PARM OPTION HAS BEEN IGNORED

DATA SET TS0045.PLIMSIM.LOAD NOW UNALLOCATED
DATA SET PLIMSIM.C1.DATA NOW UNALLOCATED
DATA SET TS0045.PLIMSIM.PLI NOW UNALLOCATED
OPTIMIZING COMPILER INVOKED
PL/1 OPTIMIZER V1 R1.1 TIME: 10.39.59  DATE: 3 JAN 73
OPTIONS SPECIFIED
 M,OBJ,F(S),OPT(TIME),NTERM
DATA SET TS0045.PLIMSIM.PLI NOW UNALLOCATED
OPTIMIZING COMPILER INVOKED
PL/1 OPTIMIZER V1 R1.1 TIME: 10.40.33  DATE: 3 JAN 73
OPTIONS SPECIFIED
 M,OBJ,F(S),OPT(TIME),NTERM
DATA SET TS0045.EVSIM.PLI NOW UNALLOCATED
DATA SET TS0045.PLIMSIM.PLI NOW UNALLOCATED
READY
```

Die Kommandoprozedur PLIML erstellt den Loadmodul:

```
exec plimsim(pliml) 'mpobj(example.obj) mainobj(example.obj(main))-
loadm(example)'

DATA SET EXAMPLE.OBJ NOW UNALLOCATED
UTILITY DATA SET NOT FREED, IS NOT ALLOCATED
DATA SET TS0045.PLIMSIM.DATA NOW UNALLOCATED
DATA SET TS0045.EVSIM.OBJ NOW UNALLOCATED
DATA SET TS0045.PLIMSIM.OBJ NOW UNALLOCATED
READY
```

Der Loadmodul wird ausgeführt, wobei das Modellprogramm MP_1 als Interrupt-Programm mit der Priorität 3 im Rechnerkern 1, das Modellprogramm MP_2 als Interrupt-Programm mit der Priorität 2 im Rechnerkern 2 ablaufen soll:

```
call example

TEMPNAME ASSUMED AS A MEMBER NAME

GIB SCHLEIFENANFANG: 1
GIB SCHLEIFENENDE: 2

GIB ANZAHL RECHNERKERNE: 2

LADE RECHNERKERN NR  1
GIB MP_NR DES INTERRUPT-PROGRAMMS: 1
GIB PRIORITAET: 3

LADE RECHNERKERN NR  2
GIB MP_NR DES INTERRUPT-PROGRAMMS: 2
GIB PRIORITAET: 2


START DES SIMULATIONSLAUFES

RECHNERKERN NR  1  'START' OR 'IDLE'?:'START'

RECHNERKERN NR  2  'START' OR 'IDLE'?:'S'


BEGIN OF SIMULATION

BEGINN: MP_2

        CLOCK    RK_NR     PROG_N      INDEX      STMT     CPUTME

            0        1          1          1         0          0
            0        1          1          1         1          0
            0        1          1          1         2          0

BEGINN: MP_1
            1        2          2          1         3          1
            1        1          1          1         3          1
            2        2          2          1         4          2
            2        1          1          1         4          2
            3        2          2          1         5          3
            3        1          1          1         5          3
            4        2          2          1         6          4
            4        1          1          1         6          4
            5        2          2          1         7          5
            5        1          1          1         7          5
            6        2          2          1         8          6
            6        1          1          1         5          6
            7        2          2          1         5          7
            7        1          1          1         6          7
            8        2          2          1         6          8
            8        1          1          1         7          8
            8        1          1          1         8          8
```

```
ENDE: MP_1

SCHLEIFENINDEX:          3
ERGEBNIS:                4
CPU-ZEIT:                8
          9        2        2        1        7        9
         10        2        2        1        8       10
         11        2        2        1        5       11
         11        2        2        1        9       11

ENDE: MP_2

SCHLEIFENINDEX:          3
ERGEBNIS:                4
CPU-ZEIT:               11

CLOCK:          11

                    AKT.ST     PAS.ST     INT.ST

RECHNERKERN NR:  1
 PROG_STAT
  PROG_NAME          1          0          1
  CPUTME             8
 REG_STAT
  PRIO               3          1          3
  AKTION             4          4          1
  IT_SPERRE          1          1          0
  IT_MASKE           0          0          0

RECHNERKERN NR:  2
 PROG_STAT
  PROG_NAME          2          0          2
  CPUTME            11
 REG_STAT
  PRIO               2          1          2
  AKTION             1          4          1
  IT_SPERRE          0          1          0
  IT_MASKE           0          0          0
          11        2          2          1       10       11
          11        2          2          1       11       11

SIMULATIONSABBRUCH

        CLOCK    RK_NR     PROG_N      INDEX     STMT     CPUTME
           11        2          2          1       11       11
END OF SIMULATION:                      11

ENDE DES SIMULATIONSLAUFES
READY
```

8.2. Darstellung eines durch PRESCAN modifizierten Modellprogramms

Das vom Preprocessor PRESCAN modifizierte Modellprogramm erhält folgende Form:

```
00010  %DCL ($N,$T) CHAR; %$N='1 '; %$T='''1''B';
00020  MP_1: PROC  ($P) RECURSIVE;
00030   %INCLUDE PRESCAN(SIMKOPF);
00040   %INCLUDE PLIMSIM(MSTDCL);  %INCLUDE EVSIM(RANDCL);
00050   %INCLUDE PRESCAN(TEST);
00060
00070  /*****************************************************************/
00080  /* AUFLOESUNG DER DO-GROUP:    Z=1;                              */
00090  /*                           DO I=A TO B;                        */
00100  /*                              Z=Z*2;                           */
00110  /*                           END;                                */
00120  /* MIT NACHGESTELLTER ABFRAGE                                    */
00130  /*****************************************************************/
00140  DCL (A,B) BIN FIXED EXT;
00150  DCL 1 STR BASED(P),
00160        2 Z BIN FIXED(31),
00170        2 I BIN FIXED(15);
00180
00190   %INCLUDE PRESCAN(INIT1);
00200   $DSA.$PF( 1)= P              ;
00210   RETURN;
00220   MP_1E:   ENTRY($P);
00230   P               =$DSA.$PF( 1);
00240   %INCLUDE PRESCAN(INIT2);
00250  $DSA.$BEW=0      ;$DSA.$LAB=$L001;$DSA.$ST=001;GOTO $RET;$L001:
00260         ALLOCATE STR;
00270  $DSA.$BEW=0      ;$DSA.$LAB=$L002;$DSA.$ST=002;GOTO $RET;$L002:
00280         PUT SKIP(2) LIST('BEGINN: MP_1');
00290     CALL $TEST('B');
00300
00310  $DSA.$BEW=1      ;$DSA.$LAB=$L003;$DSA.$ST=003;GOTO $RET;$L003:
00320         Z=1;
00330  $DSA.$BEW=1      ;$DSA.$LAB=$L004;$DSA.$ST=004;GOTO $RET;$L004:
00340         I=A;
00350     M:
00360
00370  $DSA.$BEW=1      ;$DSA.$LAB=$L005;$DSA.$ST=005;GOTO $RET;$L005:
00380         Z=Z*2;
00390  $DSA.$BEW=1      ;$DSA.$LAB=$L006;$DSA.$ST=006;GOTO $RET;$L006:
00400         I=I+1;
00410  $DSA.$BEW=1      ;$DSA.$LAB=$L007;$DSA.$ST=007;GOTO $RET;$L007:
00420         IF I<=B THEN GOTO M;
00430
00440  $DSA.$BEW=0      ;$DSA.$LAB=$L008;$DSA.$ST=008;GOTO $RET;$L008:
00450         CALL DRUCK;
00460     CALL $IDLE;
00470  $DSA.$BEW=0      ;$DSA.$LAB=$L009;$DSA.$ST=009;GOTO $RET;$L009:
00480     CALL $TEST('E');
00490
00500   GOTO $FIN;
00510  DRUCK: PROC;
00520  PUT SKIP(2) LIST('ENDE: MP_1');
00530  PUT SKIP(2) EDIT('SCHLEIFENINDEX:',I)(A(15),F(10));
00540  PUT SKIP EDIT('ERGEBNIS:',Z)(A(15),F(10));
00550  PUT SKIP EDIT('CPU-ZEIT:', $CPUTME)(A(15),F(10));
00560  END DRUCK;
00570
00580  $DSA.$BEW=0      ;$DSA.$LAB=$L010;$DSA.$ST=010;GOTO $RET;$L010:
00590
00600          GOTO $FIN ;
00610         END MP_1;
```

Der PL/1 Makro-Prozessor erzeugt daraus das folgende Source-Deck, auf dessen Statementnummern sich die System-Fehlernachrichten beziehen:

SOURCE LISTING

```
STMT

   1  MP_1: PROC  ($P) RECURSIVE;
      /*****************************************************************/
      /*  INCLUDE-DATEI: SIMKOPF                                       */
      /*  ENTHAELT DEKLARATIONEN, DIE IN MODELLPROGRAMMEN EINGEFUEGT WERDEN:*/
      /*    1. DEKLARATION DER STATISCHEN-SAVE-AREA (STSA)             */
      /*    2. DEKLARATION DER DYNAMISCHEN-SAVE-AREA (DSA)             */
      /*    3. DEKLARATION FUER DIE PROGRAMMINITIALISIERUNG            */
      /*  DER ZUGEHOERIGE FILE-NAME: PRESCAN                           */
      /*****************************************************************/
   2            DCL
                1 $STSA          STATIC,
                2 $LAB           LABEL,
                2 $PHSA          POINTER,
                2 $PARM          POINTER,
                2 $NAME          BIN FIXED       INIT(1 ),
                2 $REW           BIN FIXED       INIT(0),
                2 $CPUZEIT       BIN FIXED(31)   INIT(0),
                2 $INDEX         BIN FIXED(31)   INIT(0),
                2 $ST            BIN FIXED(31)   INIT(0),
                2 $PS_START      BIN FIXED(31)   INIT(0),
                2 $PS_REST       BIN FIXED(31)   INIT(0),
                2 $PROG_ST,
                3 $FINE          BIT(1)          INIT('0'B),
                3 $PSEUDO        BIT(1)          INIT('0'B),
                3 $TEST          BIT(1)          INIT('1'B),
                2 $PF(10)        POINTER;

   3  DCL 1 $DSA BASED($P),      /* DYN.SAVE-AREA DES MODELLPROGRAMMS  */
           2 $LAB       LABEL,          /* ADRESSE AKT.PLIM-INSTR.    */
           2 $PHSA      POINTER,        /* ADRESSE DSA DES AUFRUF.MP  */
           2 $PARM      POINTER,        /* PARAMETER-AELAGEADRESSE    */
           2 $NAME      BIN FIXED,      /* NUMMER DES MODELLPROGRAMMS */
           2 $REW       BIN FIXED,      /* AUSFUEHRUNGSDAUER PLIM-INST*/
           2 $CPUZEIT   BIN FIXED(31),  /* RECHENZEIT PROGRAMM-FOLGE  */
           2 $INDEX     BIN FIXED(31),  /* INITIALISIERUNGSZAEHLER    */
           2 $ST        BIN FIXED(31),  /* NUMMER DER PLIM-INSTR.     */
           2 $PS_START  BIN FIXED(31),  /* BEGINN PSEUDO-AKTIV-PHASE  */
           2 $PS_REST   BIN FIXED(31),  /* REST PSEUDO-AKTIV-PHASE    */
           2 $PROG_ST,                  /* PROGRAMMSTATUS             */
             3 $FINE      BIT(1),       /* ENDE-FLAGGE                */
             3 $PSEUDO    BIT(1),       /* PSEUDO-AKTIV-FLAGGE        */
             3 $TEST      BIT(1),       /* PROGRAMM-TEST-FLAGGE       */
           2 $PF(10)    POINTER;        /* SAVE-AREA PROGR.-POINTER   */

   4            DCL
                1 $LSTP          STATIC,
```

```
STMT

                2 $LL1          LABEL,
                $Q              POINTER         STATIC,
               1 $IQ_ST BASED($Q),
               2 $IQ_B  BIN FIXED(31),
               2 $IQ    BIN FIXED(31),
               1 $IP_ST BASED($P),
               2 $IP_B  BIN FIXED(31),
               2 $IP    BIN FIXED(31),
                $P              POINTER,
                $INIT           BIT(1) STATIC  INIT('0'B),
                $I              BIN FIXED(31)  STATIC,
                ADDR            BUILTIN,
                NULL            BUILTIN;
5               $Q = ADDR($LSTR);
6   DCL
    $CALL    ENTRY(BIN FIXED(31)),              /*PROCNAME*/
    $CPUTME  ENTRY RETURNS(BIN FIXED(31)),
    $DISABL  ENTRY,
    $ENABL   ENTRY,
    $IDLE    ENTRY,
    $INT     ENTRY(BIN FIXED(31),BIN FIXED,BIN FIXED,POINTER),
             /*ZEIT,MOTIV,PRIO,IT_ADR*/
    $LOAD_P  ENTRY(POINTER),                    /*PARMADDR*/
    $LOCK    ENTRY(POINTER),                    /*LOCKVARIABLE*/
    $MASK    ENTRY(BIN FIXED),                  /*MASKE (0|1)*/
    $IT_MOT  ENTRY RETURNS(BIN FIXED),          /*INT.MOTIV*/
    $RESUME  ENTRY(POINTER),                    /*ADDR.RKSTATUS*/
    $RETURN  ENTRY,
    $JUMP    ENTRY(BIN FIXED(31)),              /*PROCNAME*/
    $RUN     ENTRY(BIN FIXED(31)),              /*ZEIT*/
    $SAVE    ENTRY(POINTER),                    /*ADDR.RKSTATUS*/
    $SIGNAL  ENTRY(BIN FIXED,BIN FIXED,BIN FIXED,POINTER),
             /*RK_NR,MOTIV,PRIO,IT_ADR*/
    $CLOCK   ENTRY RETURNS(BIN FIXED(31)),
    $TEST    ENTRY(CHAR(1)),                    /*'R' | 'F'*/
    $RKDUMP  ENTRY,
    $SVC     ENTRY(POINTER),                    /*IT_ADR  */
    $STOP    ENTRY,
    $STOR_P  ENTRY(POINTER),                    /*PARMADDR*/
    $UNLOCK  ENTRY(POINTER),                    /*LOCKVARIABLE*/
    $IT_ADR  ENTRY RETURNS(POINTER),
    $CPU_ID  ENTRY RETURNS(BIN FIXED);
7            DCL $R_EP0 ENTRY RETURNS(POINTER),
                 $R_EP1 ENTRY(BIN FIXED(31)) RETURNS(POINTER),
                 $R_EP2 ENTRY(BIN FLOAT) RETURNS(POINTER),
                 $R_EP3 ENTRY(BIN FIXED(31),BIN FLOAT) RETURNS(POINTER),
                 $R_EP4 ENTRY(BIN FLOAT,BIN FIXED(31)) RETURNS(POINTER),
                 $R_EP5 ENTRY(BIN FLOAT,BIN FLOAT) RETURNS(POINTER);
8            DCL  $P_CRT  GENERIC
```

```
STMT

                          ($R_EP0 WHEN()  ,
                          $R_EP1 WHEN(BIN FIXED(31)),
                          $R_EP2 WHEN(BIN FLOAT),
                          $R_EP3 WHEN(BIN FIXED(31),BIN FLOAT),
                          $R_EP4 WHEN(BIN FLOAT,BIN FIXED(31)),
                          $R_EP5 WHEN(BIN FLOAT,BIN FLOAT)),
                 $R_DLTE  ENTRY(POINTER),
                 $R_SAVE  ENTRY(POINTER)  RETURNS(BIN FIXED(31)),
                 $R_RSET  ENTRY(POINTER,BIN FIXED(31)),
                 $RANDI   ENTRY(POINTER)  RETURNS(BIN FIXED(31)),
                 $RANDE   ENTRY(POINTER)  RETURNS(BIN FLOAT),
                 $RADIS   ENTRY(POINTER)  RETURNS(BIN FIXED(31)),
                 $BINOM   ENTRY(POINTER)  RETURNS(BIN FIXED(31)),
                 $POISS   ENTRY(POINTER)  RETURNS(BIN FIXED(31)),
                 $EXPON   ENTRY(POINTER)  RETURNS(BIN FLOAT),
                 $ERLANG  ENTRY(POINTER)  RETURNS(BIN FLOAT),
                 $GAUSS   ENTRY(POINTER)  RETURNS(BIN FLOAT);
 9       ON ERROR CALL $STOP ;

     /*****************************************************************/
     /* AUFLOESUNG DER DO-GROUP:    Z=1 ;                             */
     /*                           DO I=A TO B;                        */
     /*                              Z=Z*2 ;                          */
     /*                           END;                                */
     /* MIT NACHGESTELLTER ABFRAGE                                    */
     /*****************************************************************/
10   DCL (A,B) BIN FIXED EXT;
11   DCL 1 STR BASED(P),
           2 Z BIN FIXED(31),
           2 I BIN FIXED(15);

     /*****************************************************************/
     /*  INCLUDE-DATEI: INIT1                                         */
     /*  SIE WIRD IN MODELLPROGRAMMEN EINGEFUEGT. DIE DARIN ENTHAL-   */
     /*  TENEN STATEMENTS DIENEN DER MODELLPROGRAMM-INITIALISIERUNG.  */
     /*  DER ZUGEHOERIGE FILE-NAME: PRESCAN                           */
     /*****************************************************************/
12             IF $INIT THEN GOTO $NONINIT;
13             $INIT='1'B;
14             DO $I=1 TO 10; $STSA.$PF($I)= NULL ; END;
17             $STSA.$LAB = $LOOP ;
18             $STSA.$PARM= NULL ;
19            $NONINIT:
               $STSA.$PHSA= $P ;
20             ALLOCATE     $DSA ;
21             $STSA.$INDEX=$STSA.$INDEX + 1;
22             $DSA = $STSA, BY NAME ;
23             RETURN ;
24             $RET:  $DSA.$CPUZEIT=$DSA.$CPUZEIT + $DSA.$BEZ ;
```

STMT

```
25    $DSA.$PF( 1)= P                ;
26    RETURN;
27    MP_1E:   ENTRY($P);
28    P                =$DSA.$PF( 1);
      /*****************************************************************/
      /* INCLUDE-DATEI: INIT2                                          */
      /* INIT2 ENTHAELT EINEN TEIL DES PLIM-PROLOGS UND DIE PROGRAMM-  */
      /* TERMINIERUNG. SIE WIRD IN EIN MODELLPROGRAMM EINGEFUEGT.      */
      /* DER ZUGEHOERIGE FILE-NAME: PRESCAN                            */
      /*****************************************************************/
29              $LL: $LL1=$LL; $IP=$IG;
31              GOTO $DSA.$LAB ;
32              $:
               $FIN:
                       $DSA.$FINF    ='1'B;
33                     $DSA.$BEW     = 0   ;
34              RETURN ;
35              $LOOO: ;
36    $DSA.$BEW=0       ;$DSA.$LAB=$L001;$DSA.$ST=001;GOTO $RET;$L001:
            ALLOCATE STR;
41    $DSA.$BEW=0       ;$DSA.$LAB=$L002;$DSA.$ST=002;GOTO $RET;$L002:
            PUT SKIP(2) LIST('BEGINN: MP_1');
46       CALL $TEST('B');

47    $DSA.$BEW=1       ;$DSA.$LAB=$L003;$DSA.$ST=003;GOTO $RET;$L003:
            Z=1;
52    $DSA.$BEW=1       ;$DSA.$LAB=$L004;$DSA.$ST=004;GOTO $RET;$L004:
            I=A;
57       M:

58    $DSA.$BEW=1       ;$DSA.$LAB=$L005;$DSA.$ST=005;GOTO $RET;$L005:
            Z=Z*2;
62    $DSA.$BEW=1       ;$DSA.$LAB=$L006;$DSA.$ST=006;GOTO $RET;$L006:
            I=I+1;
67    $DSA.$BEW=1       ;$DSA.$LAB=$L007;$DSA.$ST=007;GOTO $RET;$L007:
71              IF I<=B THEN GOTO M;

72    $DSA.$BEW=0       ;$DSA.$LAB=$L008;$DSA.$ST=008;GOTO $RET;$L008:
            CALL DRUCK;
77       CALL $IDLE;
78    $DSA.$BEW=0       ;$DSA.$LAB=$L009;$DSA.$ST=009;GOTO $RET;$L009:
         CALL $TEST('E');

83     GOTO $FIN;
84    DRUCK: PROC;
85    PUT SKIP(2) LIST('ENDE: MP_1');
86    PUT SKIP(2) EDIT('SCHLEIFENINDEX:',I)(A(15),F(10));
87    PUT SKIP EDIT('ERGEBNIS:',Z)(A(15),F(10));
88    PUT SKIP EDIT('CPU-ZEIT:', $CPUTME)(A(15),F(10));
```

```
STMT

   90   END CBLOCK;

   60   $DSA.$BRK=0          ;$DSA.$LAB=$L010;$DSA.$ST=010;GOTO $RET;$L010:

                GOTO $FIN ;
   55           END MP_1;
```

Literatur:

/ 1 / J. Nehmer, D. Hilse, M. Rupp
Ein PL/1-Unterprogrammpaket für die diskrete Eventsimulation
KFK 1711, November 1972

/ 2 / D. Hilse, M. Rupp, J. Nehmer
Programmbeschreibung eines diskreten Eventsimulationssystems in PL/1
Externer Bericht 13/72-3, Dezember 1972

/ 3 / IBM System/360 Operating System
PL/1(F) - Language Reference Manual
Bestell-Nr.: GC28-8201

/ 4 / G. Fleck, M. Rupp, J. Nehmer, R. Friehmelt, D. Hilse
Programmbeschreibung des Simulationssystems PLIM
Externer Bericht 13/72-4, Dezember 1972

/ 5 / J.B. Dennis, E.C. Van Horn
Programming Semantics for Multiprogrammed Computations
Communications of the ACM, Vol.9, No.3, März 1966

Tabelle 1

Diskrete/quasistetige Zufallszahlentransformatoren

Verteilung	Name	Attribute p1	Attribute p2	Bedeutung der Parameter p1	Bedeutung der Parameter p2	Gültigkeitsbereich	Erwartungswerte Mittelwert	Erwartungswerte Varianz
Gleich-	RANDI	-	-	-	-	[1,m]	m/2	$m^2/12$ (m=2**31-2)
Gleich-	RADIS	FI	-	Obere Intervallgrenze	-	[0,p1]	p1/2	$(1+2/p1)*(p1)^2/12$
Binomial-	BINOM	FI	FL	Umfang	Einzelwahrsch.	[0,p1]	p1*p2	p1*p2*(1-p2)
Poisson-	POISS	FL	-	Mittelwert	-	[0,∞)	p1	p1
Gleich-	RANDF	-	-	-	-	(0,1)	1/2	1/12
Exponential-	EXPON	FL	-	Kehrwert des Mittelwertes	-	[0,∞)	1/p1	$1/(p1)^2$
Erlang-	ERLANG	FL	FI	Kehrwert des Mittelwertes	Ordnung	[0,∞)	1/p1	$1/((p1)^2*p2)$
Gauß-	GAUSS	FL	FL	Mittelwert	Standardabw.	(-∞,+∞)	p1	$(p2)^2$

Tabelle 2

Hilfsfunktionen
zum Betrieb des Zufallszahlentransformators (ZZT) 'name'

Funktion	Aufruf	Bedeutung
Erzeugung	pointer = $$R_CRT [(p1[,p2])]	Generic function.'pointer' vom Typ "POINTER". Richtet ZZT 'name' mit dem Identifikationsmerkmal 'pointer' ein. Bezugszufallszahl des Basisgenerators = 9699691
Aufruf	zz = $$name(pointer)	Funktionsprozedur.'zz' vom Typ "BIN FIXED(31)" bzw. "BIN FLOAT", entspr. der diskreten bzw. kontinuierlichen Verteilung der durch den ZZT'name' erzeugten Zufallszahlen.
Deaktivierung	int = $$R_SAVE(pointer)	Funktionsprozedur.'int' vom Typ "BIN FIXED(31)". Liefert die aktuelle Bezugszufallszahl des Basisgenerators.
Aktivierung	$$R_RSET(pointer,int)	Eigentliche Prozedur. Setzt die Bezugszufallszahl des Basisgenerators auf den Wert der Variablen 'int'.
Löschen	$$R_DLTE(pointer)	Eigentliche Prozedur. Löscht den ZZT mit dem ID-Merkmal 'pointer'.