Interner Bericht
DESY F58-79/01
März 1979

2585

"SIMULATION EINES PROZESSORSCHALTWERKES MIT APLSV"

von

Hans Dilcher

"Simulation eines Prozessorschaltwerkes mit APLSV"

Gliederung:

1. APL-Sprache, Bemerkungen

Der vorliegende Bericht gibt einige Erfahrungen mit der Programmiersprache APL wieder. Sie steht seit einiger Zeit im DESY-Rechenzentrum in der Variante SV-APL zur Verfügung. Die weiter unten geschilderten Arbeiten wurden jedoch schon früher am APL-SV System im Rechenzentrum der IBM-Hamburg ausgeführt.

APL:

Das APL ist eine Interpreter- und keine Kompiler-Sprache; die Programme werden als Programm-Texte (Quellen) abgelegt. Es gibt im Prinzip keine Möglichkeit, einen einmal übersetzten Befehl für später wiederholte Exekutionen abzulegen. Das System übersetzt jedesmal neu, d. h. es erzeugt den Befehlskode, vereinbart und referiert Variablen und exekutiert dann. Ein von einem Kompiler erzeugtes Programm wird darum in kürzerer Zeit exekutiert, denn Objektkode, Beschreibung und Referierung der Variablen werden vor dem Programmstart einmal angelegt. Der Anwender muß deshalb die Beschreibungen seiner Variablen dem Kompiler über besondere Deklarationen bekannt geben, und zur Laufzeit können diese Beschreibungen nur wenig geändert werden.

Wenn man z. B. Matrizen verwenden will, dann muß man deren Größe bei Kompilersprachen vorher kennen und danach die Deklarationen spezifizieren.

Der Matrix A soll in dem Befehl

$$A = B + C$$

ein Wert zugewiesen werden. Das bedeutet, daß man jedem Element a_{ij} von A den Wert $b_{ij} + c_{ij}$ zuweist. Gleichgültig, auf welche Weise nun die Operation ausgeführt wird - im FORTRAN etwa durch CALL GMADD (...) -, in jedem Fall muß die Größe von A vorher vereinbart werden (im FORTRAN: DIMENSION A (1Ø, 2Ø)).

Ein Interpreter-System hat solche Probleme nicht: Da die Variablenbeschreibungen u. U. bei jedem Befehl neu erzeugt werden, hat das System die Möglichkeit, sie jeweils nach Bedarf zu ändern und dem Benutzer die Deklarationen ganz abzunehmen. Wenn hier der Befehl A = B + C vorkommt, so richtet das System eine für B + C passende Variable ein und gibt ihr den Namen A.

Z. B.: A und B seien zwei Vektoren, dann führt ihr inneres Produkt A. B auf einen Skalar, dem man wieder den Namen A geben kann:

(1) $A \leftarrow A.B$

Der ursprüngliche Vektor A ist damit vergessen. A ist jetzt ein Skalar.

APL ist eine interpretative Sprache, u. a. mit dynamischer Variablen-Vereinbarung. Sie ist hier im DESY bei R1 implementiert (APL: A Programming Language).

Im Folgenden soll von Erfahrungen berichtet werden, die bei der Studie eines Prozessor-Entwurfs in einem APL-SV System gesammelt wurden.

Der Entwurf betrifft einen Hardware-Prozessor, also ein Schaltwerk mit Eigenschaften, die denen der Zentraleinheit einer universellen Rechenanlage gleichen.

(2)
```
IF A OP1 B TRUE
        THEN DO F: = C OP2 D OD;
FI;
```

Formel 2 stellt den in ALGOL 68 bzw. SL3-Notation geschriebenen Befehl dar, den der Hardware-Prozessor ausführt. TRUE ist für die Operationen OP1 (vgl. Abb. 1) mit " $\neq 0$ " erklärt.

Das Ziel der Studien war, die Eignung des Prozessor-Entwurfs für spezielle Programme in der On-line-Datenreduktion bei Experimenten der Hochenergie-Physik zu verwenden. In diesem Bericht soll jedoch nur von den Erfahrungen mit APL die Rede sein.

2. Beschreibung des Prozessorschaltwerkes

Abbildung 1 zeigt den Vorrat der Operationen, der in Formel 2 mit "OP1"/"OP2" gemeint ist. A, B, C, D und F bezeichnen Adressen im Datenspeicher, der aus acht Seiten mit je 32 Wörtern à 16 Bit aufgebaut ist. Die letzte der acht Seiten enthält in ihren letzten 16 Adressen spezielle Kommunikationsregister für interne und externe Zwecke: Befehlszählwerk, Ein/Ausgaberegister, Prozessorstatusregister. Alle diese Register werden von dem Befehl in Formel 1 wie die Plätze des übrigen Datenspeichers angesprochen, sind also ladbar und lesbar.

Das Mikroprogramm befindet sich in einem 48Bit breiten und vom Datenspeicher getrennten Programmspeicher. Er wird von einem Steuerrechner über ein spezielles Steuerwerk geladen.

Die beiden identischen arithmetisch-logischen Schaltwerke (ALU = arithmetic-logic-unit) sind über insgesamt 5 Datenpfade mit den Seiten des Datenspeichers verbunden. Der aus dem Programmspeicher angebotene Befehl steuert über ein Adressdekodierwerk die Seiten des Datenspeichers an. Sie bieten dann ihrerseits die Inhalte der im Befehl spezifizierten Adressen den ALU-Schaltwerken an. Außerdem bietet der Befehl den ALU-Schaltwerken ihre Operationskodes OP1 bzw. OP2 an.

Der Ausgang von ALU B entscheidet, ob das an ALU A anstehende Operationsergebnis in die Adresse F oder die Adresse Null des Datenspeichers zurückgeschrieben wird. Das geschieht durch das Signal IF der ALU B, mit dem die Adresse F gelöscht wird. Auf diese Weise wird die IF-Bedingung im Befehl (Formel 2) verwirklicht.

Anmerkung 1:
In dem in Abbildung 2 aufgezeichneten Prozessor ist der Mikrobefehl mit 48 Bit angegeben, bisher wurden aber nur 46 Bit erklärt (5 mal 8-Bit-Adressen und 2 mal 3-Bit-Operationskodes). Mit den restlichen beiden Bits geschieht Folgendes:

1. Eines der beiden Bits kann die TRUE/FALSE-Abfrage in der Nebenbedingung in Formel 2 umkehren. Der IF-Ausgang der ALU wird umgekehrt.
2. Das andere Bit findet als Kettungsbit bei Operationen mit doppelter oder mehrfacher Wortlänge Verwendung: Bei jeder ALU-Operation wird der Zustand

des CARRY-OUT Signales der ALU (z. B. MECL 10181) in einem Flip-Flop gespeichert. Ist nun in einem Befehl das Kettungsbit gesetzt, so wird jetzt der Speicherflipflop mit dem CARRY-OUT auf die CARRY-IN Leitung der ALU gegeben. Bei Operationen mit mehrfacher Wortlänge (z. B. 32 Bit) muß man für jedes 16 Bit-Teilwort einen Befehl schreiben, aber das Kettungsbit ermöglicht die Verwaltung des Obertrages. Es kann außerdem in einem in Seite 7 des Datenspeichers enthaltenen Prozessorstatusregister vom Programm abgefragt werden.

3. Arbeiten mit APLSV

3.1. Programmiersystem für das Prozessorschaltwerk

Abbildung 3 zeigt das Flußdiagramm der Funktion PROGECLP. Formel 3 zeigt ihren Aufruf. Die Programmquelle wird entsprechend dem APL-System im Dialog eingegeben.

(3) PROG ← PROGECLP

Der erzeugte Objektkode wird der Variablen PROG zugewiesen. Er wird jeweils nach Eingabe einer Quellzeile erzeugt. Das erspart das Umsetzen der gesamten Quelle bei Programmänderungen, denn PROGECLP verbraucht merklich CPU-Zeit im APL-System. Außerdem ist dann eine Syntaxanalyse des eingegebenen Befehls sofort möglich.

PROGECLP verwendet im Einzelnen folgende Funktionen:
INPUT ermöglicht die Eingabe einer Folge von APL-Symbolen und erzeugt daraus einen Vektor, dessen Elemente die entsprechenden Indizes der Systemvariablen ALLE ZEICHEN sind.

SEPRTE erzeugt daraus eine Matrix, indem sie den Vektor nach Maßgabe eines Trennzeichens (hier der Zwischenraum = SPACE) aufteilt, alle so entstandenen Teilvektoren mit Nullen auf gleiche Länge auffüllt und zu einer Matrix zusammensetzt.

UPDATE führt die Syntaxanalyse durch und erzeugt den Objektkode für die eingegebene Quellzeile.

NEWPR wird zu Beginn einer Programmeingabe aufgerufen und richtet einige später benötigte Variablen ein.

PROT erzeugt ein Protokoll der Quelle in der Variablen SOURCE. OUT druckt die gesamte Quelle aus. Diese Funktionen werden über Steuerzeichen vom Benutzer aufgerufen.

Die Eingabe einer Folge von Quellzeichen wird immer durch die Eingabe von ENDE am Anfang einer Zeile beendet. PROGECLP erwartet dann ein neues Steuerzeichen vom Benutzer.

Von den in PROGECLP aufgerufenen Funktionen sind nur NEWPR und UPDATE auf die speziellen Prozessoreigenschaften abgestimmt. Will man einen anderen Prozessor simulieren, brauchen neben der Funktion ECLP, die den Prozessor selbst simuliert, nur diese beiden Funktionen verändert zu werden. Insoweit ist PROGECLP also allgemein für die Simulation von Prozessorschaltwerken verwendbar.

3.2. Simulation des Prozessorschaltwerkes

Abbildung 4 zeigt das Flußdiagramm der Funktion ECLP, die das in Abbildung 2 beschriebene Schaltwerk simuliert. Formel 4 zeigt den Aufruf.

(4) RSLT ← PROG ECLP DATA

PROG, der Name ist frei wählbar, ist zuvor von PROGECLP erzeugt worden. Die Variable DATA enthält den Datenspeicher des Prozessors. Sie ist durch PROGECLP festgelegt, da dort einige Unterfunktionen zugreifen. In ECLP liegt der Name DATA dagegen nicht fest. Die durch den Prozessor zu verarbeitenden Daten werden unter PROGECLP in DATA eingetragen. Die Variable RSLT, der Name ist frei wählbar, enthält den Datenspeicher nach der Exekution von ECLP.

Der Prozessor hält an, wenn entweder eine voreingestellte Anzahl von Befehlen exekutiert wurde, oder wenn das Programm auf Adresse Null verzweigt. Nach dem Prozessorhalt gibt ECLP einen Vektor aus, dessen Elemente die Nummern der aufgeführten Befehle in der zeitlichen Reihenfolge enthalten.

Die Konsolprotokolle für eine Anwendung von PROGECLP und ECLP werden in Abbildung 5 und 6 gezeigt.

3. Bemerkungen zu APL:

Unter den zur Verfügung stehenden Sprachen (FORTRAN, PL1, Assemblersprachen, APL) erschien APL die hier am besten geeignete Sprache zu sein: Ein interpretatives System hat den Vorzug, die für diese Arbeiten sehr wünschenswerten Dialogeigenschaften bereits zu enthalten. In der Sprache APL ist der Benutzer frei von aller Verpflichtung zur Deklaration und Dimensionierung von Variablen und Variablentypen. Speziell diese Eigenschaft reduziert die Programmierzeit gegenüber der Anwendung anderer Programmiersprachen erheblich. Da bei den hier geschilderten Studien die Lebensdauer der Programme, das heißt die Zeit zwischen zwei Programmänderungen kurz ist im Vergleich zum gesamten Zeitaufwand, ist die Programmierzeit der ausschlaggebende Gesichtspunkt für die Verwendung eines APL-Systems gewesen.

Für die hier beschriebenen Arbeiten wurden 21 Minuten CPU-Zeit und etwa 50 Stunden Konsolzeit an einem IBM-System 370/158 benötigt.

3.3. Ein Beispiel zur Benutzung von Programmiersystem und Prozessor

Abbildung 6 zeigt das Konsolprotokoll einer Programmierung des Prozessors mit Hilfe der Funktion PROGECLP. Das Programmiersystem wird mit einem Aufruf nach Formel 2 gestartet. Das Zeichen ⍞ zeigt an, daß PROGECLP eine Eingabe erwartet. Es wird das Zeichen v"o" für den Aufruf von NEWPR eingegeben. PROGECLP fordert zur Eingabe des Programmnamens auf. Danach werden mit Hilfe von Steuerzeichen an PROGECLP einige Variable vereinbart und mit Werten belegt. Nach jedem Dialog erwartet PROGECLP wieder ein Steuerzeichen. Mit → wird die eigentliche Eingabe des Programms gestartet. Sie wird durch die Zeichen "ENDE" am Anfang einer Zeile abgeschlossen. PROGECLP erwartet wieder Steuerzeichen. In Abbildung 5 wurde ☐ eingegeben. Damit druckt PROGECLP die gesamte Quelle aus. Mit ∈ wurde PROGECLP verlassen.

Das eingegebene Programm führt folgenden Algoritmus aus: In einer Schleife wird der Inhalt des Platzes A im Datenspeicher von Null auf 10 erhöht. Der Endwert ist in der Variablen K2 abgelegt und wird jedesmal abgefragt. Ist er erreicht, so wird nach Programmadresse Null (in der Variablen K0) verzweigt. Das bedeutet den Halt des Prozessors. Die Befehle 2 und 3 werden dabei wegen des Instruction-Pipelining immer durchlaufen. Bei komplexeren Programmen muß

der Benutzer darauf achten, auch den Befehl 3 mit sinnvollen Anweisungen zu formulieren, da sonst die Exekutionszeit des Prozessors von 60ns pro Befehl nicht erreicht wird.

Anmerkung:

Man hätte das Schaltwerk des Prozessors auch so einrichten können, daß die Instruction-Pipeline bei Verzweigungen abgebrochen wird. Das hätte aber das Schaltwerk komplizierter werden lassen. Bezüglich der Exekutionsgeschwindigkeiten sind beide Verfahren gleichwertig. Wenn man die Befehle 2 und 3 für sinnvolle Anweisungen nutzen kann, dann hat der vermiedene Abbruch der Pipeline sogar den Vorteil, daß die Exekutionszeit verkürzt wird.

Abbildung 6 zeigt ein Konsolprotokoll mit der Prozessorsimulation durch die Funktion ECLP. Die dem Funktionsaufruf folgenden vier Zeilen werden von ECLP ausgegeben. Die letzte der Zeilen zeigt einen Vektor, der in seinen Elementen die Nummern der vom Prozessor exekutierten Befehle in der Reihenfolge ihrer Ausführungen aufführt. Das Ergebnis der Rechnung wird nicht ausgegeben. Es muß einem Ausdruck der Variablen RSLT entnommen werden.

Die Funktion PROGECLP benötigte in diesem Beispiel 15 Sekunden CPU-Zeit auf einer 370/158, die Funktion ECLP benötigte vier Sekunden.

```
      PROG←PROGECLP
●
o
PROGRAMMNAMEN EINGEBEN
●
22.8.1978 DEMO
●
T
BITTE DEN NAMEN EINGEBEN
●
KO
BITTE DEN WERT EINGEBEN
O:
     0
●
T
BITTE DEN NAMEN EINGEBEN
●
K1
BITTE DEN WERT EINGEBEN
O:
     1
●
T
BITTE DEN NAMEN EINGEBEN
●
K2
BITTE DEN WERT EINGEBEN
O:
     10
●
T
ADRESSWERT EINGEBEN
O:
     20
```

cont.

Operationskode: OP1/OP2	Funktion:
Ø	F: = A
1	F: = $\overline{A}$
2	F: = A * 2
3	F: = A : 2
4	F: = A + B
5	F: = A - B
6	F: = A ∧ B
7	F: = A ∨ B

Abbildung 1: Operationen der Operatoren OP1 und OP2

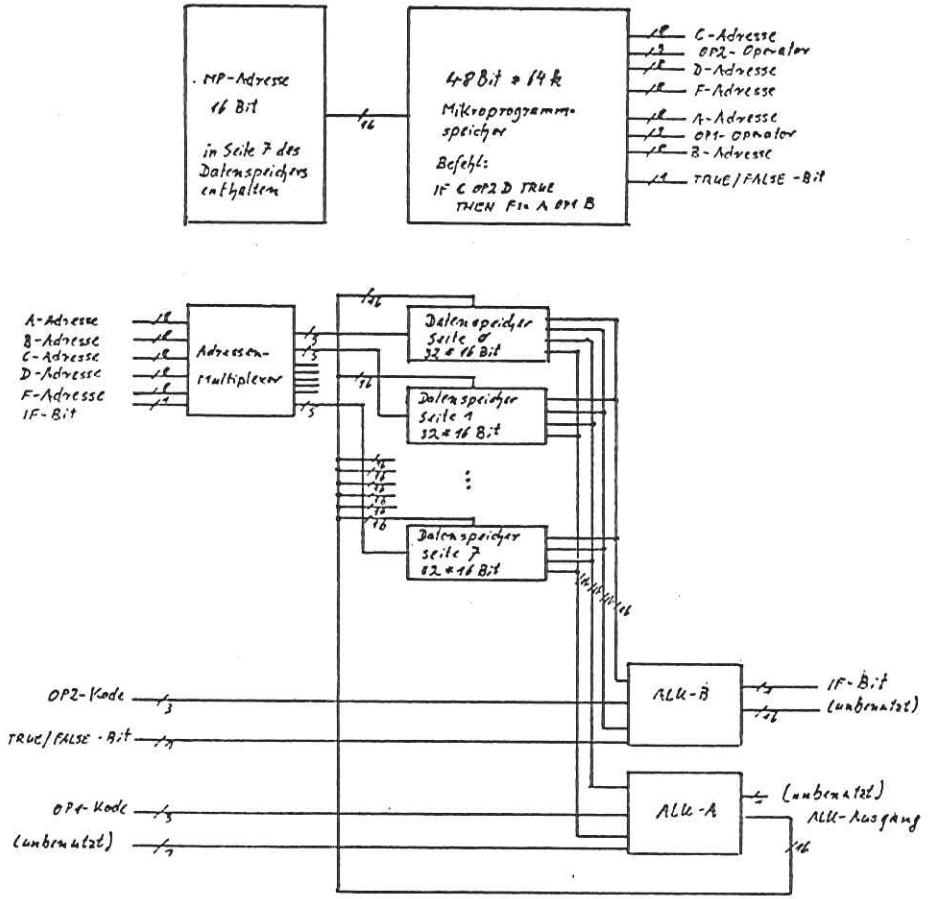

Abbildung 2: Blockschaltbild des Prozessorschaltwerkes

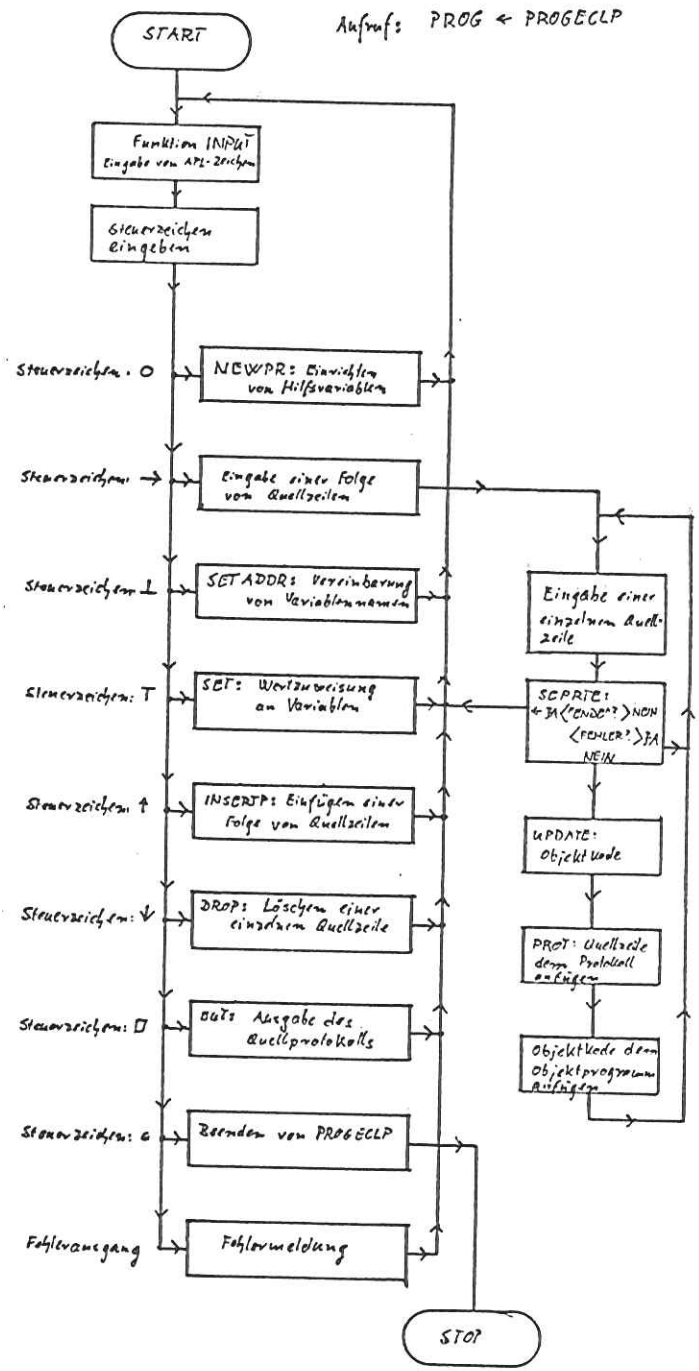

Abbildung 3: Flußdiagramm der Programmierfunktion PROGECLP.
Der Programmabdruck ist in Abbildung 7 gezeigt

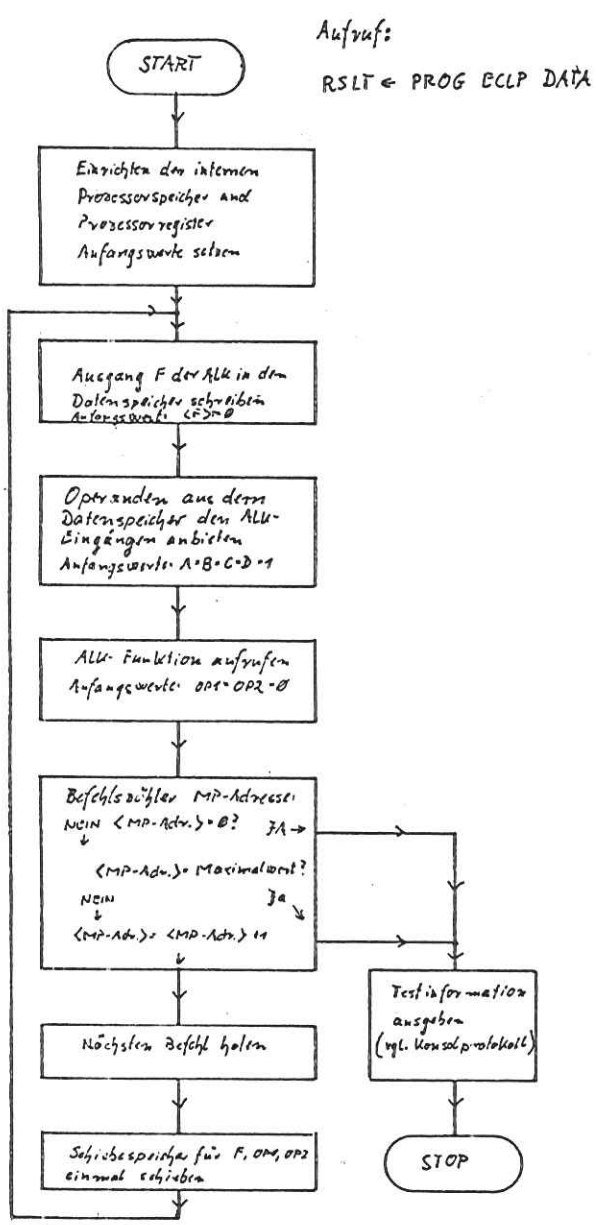

Abbildung 4: Flußdiagramm der Simulationsfunktion ECLP, die das in Abbildung 2 gezeigte Schaltwerk simuliert

```
ADRESSNAMEN EINGEBEN
●
A
●
→
●
LOOP : IF A - K2 TRUE THEN MPC = LOOP
●
A = A + K1
●
B = A
●
MPC = KO
●
A = A
●
A = A
●
A = A
●
ENDE
DAS WAR DAS ENDE DER PROGRAMMEINGABE
●
□
* 0  22.8.1978 DEMO
* 1  LOOP : IF A - K2 TRUE THEN MPC = LOOP
* 2  A = A + K1
* 3  B = A
* 4  MPC = KO
* 5  A = A
* 6  A = A
* 7  A = A
●
```

PROGRAMMSCHLEIFE MPC IST DER NAME DER MIKROPROGRAMMADRESSE (IM DATENSPEICHER)

Abb.5:
Konsolprotokoll einer Programmeingabe mit PROGECLP und Ausdruck des eingegebenen Programms.

```
      RSLT←PROG ECLP DATA
PROCESSOR HALT.
ANZAHL DER AUSGEFUEHRTEN INSTRUKTIONEN:
37
0 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 4 5
```

Abb.6:
Konsolprotokoll eines ECLP-Simulationsverlauf.

```
      ▽PROGECLP[□]▽


      ▽ O←PROGECLP;A;B;C;D;□IO
[1]   ⍝DIESE FUNKTION  ERZEUGT DEN OBJEKTCODE FUER DEN ECLP
[2]    □IO←1
[3]   STRT:INSTR←(INPUT),0
[4]   ⍝VERZWEIGUNGEN F←R DIE VERSCHIEDENEN EINGABE-BETRIEBSARTEN.
[5]    →(INSTR[1]=187)/OUTPUT
[6]    →(INSTR[1]=183)/INSRT
[7]    →(INSTR[1]=184)/DRP
[8]    →(INSTR[1]=185)/START
[9]    →(INSTR[1]=149)/NEWPROG
[10]   →(INSTR[1]=168)/END
[11]   →(INSTR[1]=170)/VARDKL
[12]   →(INSTR[1]=169)/ADDRDKL
[13]   □←'STEUERZEICHEN FEHLT'
[14]   →STRT
[15]  START:INSTR←INPUT
[16]   NAMES←SEPRTE INSTR
[17]   →(ERRFLAG=1)/START
[18]   →(ENDFLAG=1)/STRT
[19]   OBJV←INSTRCNT UPDATE NAMES
[20]   →(ERRFLAG=1)/START
[21]   SOURCE←SOURCE PROT INSTR
[22]   OBJF←OBJF,[1] OBJV
[23]   INSTRCNT←INSTRCNT+1
[24]   →(PRGL≥INSTRCNT)/START
[25]   □←'DER PROGRAMMSPEICHER IST VOLL.PROGAMMEINGABE MIT  E N D E  ABSCHLIESSEN.
[26]   →START
[27]  ⍝STEUERZEICHEN FUER VARIABLEN DEKLARATION:⊤
[28]  VARDKL:SET
[29]   →STRT
[30]  ⍝STEUERZEICHEN FUER ADRESSVEREINBARUNGEN:⊥
[31]  ADDRDKL:BERTA←SETADDR BERTA
[32]   →STRT
[33]  ⍝STEUERZEUCHEN FUER AUSGABE:□
[34]  OUTPUT:OUT SOURCE
[35]   →STRT
[36]  ⍝STEUERZEICHEN FUER EINFUEGEN VON BEFEHLEN:↑
[37]  INSRT:SOURCE←INSERTP SOURCE
[38]   →STRT
[39]  ⍝STEUERZEICHEN FUER DAS LOESCHEN VON BEFEHLEN:↓
[40]  DRP:SOURCE←DROP SOURCE
[41]   →STRT
[42]  ⍝STEUEZEICHEN FUER DAS FORTETZEN DER EINGABE:→
[43]  CONT:→START
[44]  NEWPROG:OBJF←NEWPR
[45]  ⍝STEUERYEICHEN FUER EINGABE EINES NEUEN PROGRAMMS:⍥
[46]   →STRT
[47]  ⍝STEUERZEICHEN FUER FLUCHT:⍷
[48]  END:0←OBJF
      ▽
```

Abbildung 7: Ausdruck der Funktion PROGECLP

```
      ▼ECLP[□]▼


      ▼ RSLT←PROG ECLP DATA;A;B;C;D;E;F;XA;XB;XC;XD;IMAX;FSHFT;DRAM;PA;PN;DIO
[1]   ⍝IMAX IST DIE MAXIMALE ANZAHL VON AUSGEFUEHRTEN INSTRUKTIONEN
[2]   ⍝IVECT IST EIN VEKTOR,DER ALS ELEMENTE FORTLAUFEND VON LINKS
[3]   ⍝NACH RECHTS DIE NUMMERN DER EXEKUTIERTEN INSTRUKTIONEN ENTHAELT
[4]    IVECT←0
[5]    IMAX←100
[6]   ⍝ EINRICHTUNG DER PROZESSORSPEICHER UND -REGISTER
[7]   ⍝ WZ IST DIE WORTANZAHL,SZ DIE SATZANZAHL DES DATENSPEICHERS DATA
[8]   ⍝ SZ UND WZ WERDEN ZU ANFANG DURCH NEWPR GESETZT
[9]    DIO←0
[10]  C←WZ-1
[11]  F←SZ-1
[12]  OPR←(2 3)⍴0
[13]  FSHFT←(2 3)⍴0
[14]  DRAM←DATA
[15]  MUEPR←PROG
[16]  MPCADR←(C,F)
[17]  PADDR← 0 3 7 10
[18]  PAGE← 1 4 8 11
[19]  OPCODE← 2 9
[20]  XA←SZ-5
[21]  XB←SZ-4
[22]  XC←SZ-3
[23]  XD←SZ-2
[24]  XA←DATA[C;XA]
[25]  XB←DATA[C;XB]
[26]  XC←DATA[C;XC]
[27]  XD←DATA[C;XD]
[28]  TF←12
[29]  C←13
[30]  F← 5 6
[31]  ALUF←1
[32]  PN←PA←5⍴0
[33]  TFB←CNCT←3⍴1
[34]  INSTR←(12⍴0),1,0
[35]  ICOUNT←1
[36] ⍝PROZESSOR-INSTRUCTION-PIPELINE
[37] NEXT:A←XC OR A←PN[0]
[38]  B←PA[0]
[39] ALOPR1:ALUA←DRAM[A;B]
[40]  A←XD OR A←PN[1]
[41]  B←PA[1]
[42]  ALUB←DRAM[A;B]
[43]  COND←ALUFKT OPR[0;1]
[44]  A←XA OR A←PN[2]
[45]  B←PA[2]
[46] ALOPR2:ALUA←DRAM[A;B]
[47]  A←XB OR A←PN[3]
[48]  B←PA[3]
[49]  ALUB←DRAM[A;B]
[50]  A←FSHFT[0;2]
[51]  B←FSHFT[1;2]
[52]  DRAM[A;B]←ALUF
[53]  ALUF←ALUFKT OPR[1;1]
[54]  →(TFB[1]=(COND≠0))/RAMSEL
[55]  FSHFT[;1]←(0,1)
[56] RAMSEL:PN←INSTR[PADDR]
[57]  PA←INSTR[PAGE]
[58]  OPR[;0]←INSTR[OPCODE]
[59]  FSHFT[;0]←INSTR[F]
[60]  TFB[0]←INSTR[TF]
[61]  CNCT[0]←INSTR[C]
[62]  FSHFT←¯1⌽FSHFT
[63]  TFB←¯1⌽TFB
[64]  OPR←¯1⌽OPR
[65]  CNCT←¯1⌽CNCT
[66]  A←1↑MPCADR
[67]  B←1↓MPCADR
[68]  →((+/DRAM[A;B])=0)/STP
[69] ⍝ STOP DURCH SPRUNG NACH PROGRAMMADRESSE 0.
[70]  DRAM[A;B]←1+D←,,DRAM[A;B]
[71]  IVECT←IVECT,D
[72]  →((1↑⍴MUEPR)≤D)/STP
[73]  INSTR←,MUEPR[D;]
[74]  ICOUNT←ICOUNT+1
[75]  →(ICOUNT≤IMAX)/NEXT
[76]  D←'INSTRUCTIONOVERFLOW.PROCESSOR STOPPED.'
[77]  →END
[78] STP:D←'PROCESSOR HALT.'
[79]  D←'ANZAHL DER AUSGEFUEHRTEN INTRUKTIONEN:'
[80]  D←ICOUNT
[81] END:RSLT←DRAM
[82]  D←IVECT
      ▼
```

Abbildung 8: Ausdruck der Funktion ECLP