Mitteilung Nr. 65

Optimierung und Erweiterung des DECSystem-10 PASCAL Compilers

B. Brügge, K. Mühle, B. Nebel,
H. Faasch und H.-H. Nagel

IfI-HH-M-65/79

Februar 1979

Mit dieser Mitteilung wird der Versuch unternommen, die in den vergangenen zwei Jahren am DECSystem-10 PASCAL Compiler durchgeführten Arbeiten wenigstens teilweise zu dokumentieren.

Die Grundlage dazu bildet ein Bericht von K. Mühle und B. Nebel über eine neue Registerallokation. Im Rahmen dieser Arbeiten wurden auch zahlreiche Kleinigkeiten beseitigt. Besonders B. Brügge hat sich darum verdient gemacht, die Fülle der inzwischen vorliegenden Hinweise zu sichten, zu sortieren, zu überprüfen und gegebenenfalls durch Compiler-Änderungen zu berücksichtigen. Die Umstellung der Terminal-Ein/Ausgabe auf die Standard-Files INPUT/OUTPUT ist im wesentlichen sein Werk. H. Faasch setzt diese Arbeiten fort. Diese Kleinarbeit - z.T. auch die Beseitigung (bisher noch) unvermeidbarer Fehler bei der Änderung des Compilers - läßt sich nur sehr schwer belegen. Eine Auswahl der Maßnahmen ist im zweiten Teil dieser Mitteilung aufgeführt, gefolgt von einer kleinen Untersuchung über Speicherbedarf und Laufzeit der Compiler-Versionen.

Die Fertigstellung dieser Mitteilung verzögerte sich, da ich lange Zeit nicht dazu kam, alle Unterlagen zusammenzutragen und durchzusehen.

Hamburg, im Februar 1979

H.-H. Nagel

Implementation eines Registerallokationsalgorithmus nach Ammann fuer den DECSystem-10 PASCAL-Compiler

Kl. Muehle und B. Nebel

1. Ausgangssituation

Im alten Compiler werden die Register von der Codeerzeugung des Compilers kellerartig verwaltet. Insbesondere wird nicht erinnert, ob ein Wert schon in ein Register geladen wurde, so dass vermeidbare Ladebefehle erzeugt werden. Zum genauen Registerbelegungsalgorithmus siehe [1].

2. Kurze Beschreibung des "Ammann-Algorithmus"

Eine Beschreibung findet sich in [2], [3]. Ammann hat in den CDC-6000 PASCAL-Compiler Datenstrukturen und ein Prozedurenpaket eingebaut, die ueber die Registerbelegung Buch fuehren. Da diese Buchfuehrung moeglichst einfach sein soll, um die Compilationszeit nicht zu stark zu belasten, werden nur Registerinhalte als erinnerungswuerdig angesehen, die sich durch maximal zwei einfache Daten beschreiben lassen. Das hat zur Folge, dass Ausdruecke, die Operatoren enthalten, nicht erinnerungswuerdig sind.

Durch die Buchfuehrung erreicht Ammann, dass, falls ein erinnerungswuerdiger Wert schon geladen ist, eine Ladeinstruktion gespart wird, oder - falls eine Nachbaradresse geladen ist -, die Ladeinstruktion sich auf eine kurze (15-Bit) Instruktion reduziert.

Um eine moeglichst reichhaltige Registerbelegung zu erhalten, werden die Register natuerlich nicht kellerartig sondern "streu-" belegt.

Speicherungsinstruktionen werden nicht verzoegert, da dieses im Fall einer Ein-Phasen-Compilation keinen grossen Gewinn bringt.

3. Aspekte, die sich bei einer Uebertragung ergeben

3.1. Maschinenbezogen

Eine "Hand"-Simulation der Codeerzeugung mit Buchfuehrung ueber die Registerbelegung fuehrte gegenueber der jetzigen Codeerzeugung durch den DECSystem-10 PASCAL-Compiler (PASRLX, Version vom 30. Dez. 1976) zu folgenden Einsparungen an Code:

3.1.1. Programm EASTERN, ein Programm zur Errechnung der Osterfeiertage, das von Ammann benutzt wurde, um die Effizienz seiner Codeerzeugung aufzuzeigen: Ersparnis 14% (bei Ammann 30%)

3.1.2. Procedure COMPTYPES aus dem PASCAL-Compiler (PASRLX.PAS): 8% Ersparnis.

3.1.3. Procedure ENTERID ebenfalls aus dem PASCAL-Compiler: 14% Ersparnis.

Diese etwas "maesren" Ergebnisse verglichen mit denen des CDC-6000 Compilers haben folgende Ursachen :

Die CDC-6000 verhaelt sich bei der Verknuepfung von Registerinhalten wie eine drei-Adressmaschine, sonst wie eine eineinhalb Adressmaschine. Insbesondere existieren keine Befehle, die Registerinhalte mit Speicherinhalten verknuepfen. Daraus folgt:

a) Fuer jeden Operanden muss eine Ladeinstruktion erzeugt werden, soweit er nicht schon im Register steht.

b) Arithmetische Operationen ueberschreiben nicht notwendigerweise einen Operanden.

Das DECSystem-10 verhaelt sich grundsaetzlich wie eine eineinhalb Adressmaschine. Das heisst:

a) Es braucht bei einer zweistelligen Operation nur ein Operand geladen werden.

b) Der geladene Wert wird durch eine arithmetische Operation mit dem Ergebnis ueberschrieben, womit der Inhalt des betroffenen Registers nicht mehr erinnerungswuerdig ist.

Im Gegensatz zur CDC-6000 besitzt das DECSystem-10 eine konstante Befehlslaenge, so dass die oben erwaehnte Codereduktion durch Kurze Befehle im DECSystem-10 Compiler nicht zum Tragen Kommt.

3.2. Compilerbezogen

3.2.1. Bytepointer

Durch die Streubelegung der Register sinkt die Wahrscheinlichkeit, verglichen mit der jetzigen Registerverwaltung, dass in den Bytepointern, die fuer die Adressierung gepackter Recordfelder benoetigt werden, das gleiche Basisregister steht, d.h. es werden mehr Bytepointer als bisher in den Code eingetragen werden.

3.2.2. Sprungketten

Falls die Idee realisiert wird, boolesche Ausdruecke als Sprungketten zu uebersetzen, kann bei den nachfolgenden Anweisungen nur von der Registerbelegung nach dem ersten Teilausdruck ausgegangen werden, d.h. dass die beiden Optimierungen gegeneinander arbeiten.

4. Fazit:

Ammann erzielte durch diese und andere Optimierungen ([2] S.119 ff) eine Codereduktion des Compiler um 25%. Diesen Wert werden wir aus den oben genannten Gruenden nicht erreichen. Unsere Erwartungen liegen bei einer Reduktion in Benutzerprogrammen von 10%. Die Codereduktion des Compilers wird bei ca. 4% liegen.

Innerhalb des CDC-6000 PASCAL Compilers haben die Registerverwaltungsprozeduren einen Laufzeitanteil von 20%, so dass wir eine Laufzeitsteigerung beim DECSystem-10 PASCAL-Compiler von ca. 10% erwarten.

5. Die Datenstrukturen fuer die Registerverwaltung in der konkreten Implementation

5.1. Es werden nicht alle 16 Register von der Registerverwaltung beruecksichtigt, sondern nur der Bereich FIRSTREG bis LASTREG (2..13). Die anderen werden fuer folgende Zwecke benoetigt:

Register 0: Ist nicht als Indexregister verwendbar und wird von der Laufzeitunterstuetzung verwendet. Um den Verwaltungsaufwand moeglichst gering zu halten, wird dieses Register nicht mit in die Optimierung einbezogen. In einer spaeteren Implementationsstufe koennte man daran denken, in der linken Haelfte von Register 0 eine 1 und in der rechten Haelfte den Wert NIL zu halten. Dadurch werden Zuweisungen von NIL und 1 bedeutend effektiver.

Register 1: Wird bei Byte-Pointer-Zugriffen auf gepackte Array-Komponenten benoetigt. Dadurch wird erreicht, dass nicht fuer jedes Register ein Satz von Byte-Pointer erzeugt wird. Um Verklemmungen zu vermeiden, haben wir davon abgesehen, dieses Register in die Optimierung mit einzubeziehen.

Register 14, 15: (NEWREG, TOPP) werden zur Verwaltung der Halde und des Laufzeitkellers benoetigt.

Wir haben die Zuordnung von NEWREG und BASIS vertauscht und BASIS mit in die Registerverwaltung aufgenommen. Dadurch wurde die Formulierung der Verwaltungsprozeduren an manchen Stellen einfacher.

5.2. Erinnerungswuerdige Registerinhalte sind

a) einfache Variablen. Das sind Variablen, die sich durch den Deklarationsspegel, Relativadresse und gefs. die Art der Packung beschreiben lassen.

b) indirekte Variablen, die entweder durch ein Indirektregister und eine Relativadresse (inirekt Bit = 0) oder durch Pegel und Relativadresse (indirekt Bit = 1) beschrieben werden.

c) Konstanten. Sie werden durch ihren Wert beschrieben.

d) Basisadressen von Prozedur-Aktivierungs-Records, die man durch den zugehoerigen Pegel beschreiben kann.

Daraus ergibt sich die Struktur von REGSTATUS, die Beschreibung eines Registerinhaltes.

5.3. REGSTATUS

```
CNTTYP = ( ONE, TWO, PREFERREDTWO );

LENGTHTYP = ONE..TWO;

REGCONTENTS = ( AVAIL, OTHER, SECOND, CNST, BASADDR, VARB );

REGCNT1 = CNST..VARB;

REGSTATUS = RECORD
                REFNR : INTEGER;
                LENGTH : LENGTHTYP;
                CASE CONTENTS : REGCONTENTS OF
                    SECOND, CNST
                    BASADDR, VARB:
                     (LASTREF : ADDRRANGE;
                      CASE REGCNT1 OF
                        CNST:    (SHRT : BOOLEAN;
                                  RDVALU : VALU);
                        BASADDR: (BLEVEL : LEVRANGE);
                        VARB:    (RDLEVEL : 0..MAXLEVEL;
                                  RDDPLMT : ADDRRANGE;
                                  RDACCESS : ACCESSKIND;
                                  REFREG : ACRANGE;
                                  INDREFNR : INTEGER;
                                  RDPACKK : PACKKIND;
                                  RDBYTE : BPOINTER;
                                  RDBPADDR : ADDRRANGE));
            END;
```

Die Bedeutung der einzelnen Felder:

a) REFNR: Der Referenzzaehler zeigt an, wie oft der Registerinhalt referiert wird, d.h. wie oft der Registerinhalt waehrend der Compilation des aktuellen Ausdrucks angefordert und noch nicht weiterverarbeitet wurde. Z.B. A[I]:=I+3 : Bei der Compilation des Ausdrucks I+3 steht der Referenzzaehler der Registerbeschreibung von I auf 2. In der

jetzigen Implementationsstufe lassen wir allerdings REFNR>1 nur bei Inhalten zu, die nicht durch arithmetische Operationen zerstoert werden koennten, z.B. Basisadressen und anonymen WITH-Variablen, da dieses die Verwaltung erheblich vereinfacht und im anderen Fall der Optimierungsgewinn wahrscheinlich nicht erheblich waere.

b) LENGTH: Falls CONTENTS <> AVAIL ist (s.u.) gibt LENGTH an, ob das Register ein 36-Bit Datum oder einen Teil eines 72-Bit langen Datums enthaelt.

c) CONTENTS kann folgende Werte annehmen:

AVAIL: Das Register ist frei verfuegbar und enthaelt keinen erinnerungswuerdigen Wert. Impliziert REFNR=0;

OTHER: Der Inhalt des Registers ist nicht erinnerungswuerdig, wird aber bei der weiteren Compilation noch gebraucht. Impliziert REFNR=1;

SECOND: Der Registerinhalt ist das zweite Wort eines Doppelwortdatums. Die naehere Spezifikation ist in diesem Fall in der Registerbeschreibung des Registers mit der naechst Kleineren Registernummer vorhanden. In diesem Fall wird in beiden Registerbeschreibungen LENGTH=TWO eingetragen. Der Referenzzaehler REFNR hat den gleichen Wert wie der Referenzzaehler des Registers, das das erste Wort enthaelt.

CNST: Der Inhalt des Registers ist eine Konstante

BASADDR: Der Inhalt ist eine Basisadresse.

VARB: Der Inhalt ist eine Variable.

d) LASTREF: Im Moment des Erloeschens der letzten Referenz wird, falls der Inhalt erinnerungswuerdig ist, LASTREF auf den Wert des Instruktionszaehlers (IC) gesetzt, um nach einem LRU-Algorithmus zu entscheiden, welches Register bei einer Anforderung (NEEDREG) ueberschrieben wird.

Eine Konstante wird durch zwei Werte beschrieben. RDVALU vom Typ VALU, sowie das Feld SHRT, welches angibt, ob das Feld IVAL oder VALP im VALU-Record gemeint ist.

Eine Basisadresse wird durch den zugehoerigen Pegel beschrieben. Dieser Wert ist in BLEVEL enthalten.

Eine erinnerungswuerdige Variable wird durch folgende Felder beschrieben: Informationen ueber die Packungsart werden in RDPACKK, RDBYTE und RDBPADDR gespeichert. RDDPLMT enthaelt bei indirekten Zugriffen, bei denen das Indirektbit nicht gesetzt ist, die Relativadresse bezueglich des Indirektregisters (REFREG), ansonsten die Relativadresse der Variablen bezueglich der Basisadresse des Prozeduraktivierungsrecord. Indirektzugriffe, bei denen das Indirektbit nicht gesetzt ist, entstehen durch Zugriffe auf durch Zeiger referierte Felder eines Records, waehrend bei Zugriffen auf unstrukturierte VAR-Parameter das Indirektbit gesetzt wird. RDLEVEL beschreibt den Deklarationspegel. RDACCESS beschreibt den Zugriff auf

die Variable. Diese Information wird benoetigt, falls die Registerbeschreibungen geaendert werden (s. COINZ_DEL, DELETEINDIRREFS). REFREG ist bei indirekten Zugriffen, bei denen das Indirektbit nicht benutzt wird, das Indirektregister, ansonsten hat REFREG den Wert 0. Da eine indirekt referenzierte Variable, bei der ein Indirektregister benutzt wird, nur solange erinnerungswuerdig ist, wie die Registerbeschreibung von REFREG nicht geloescht wird, sollten Indirektregister moeglichst nicht geloescht werden. Dieses erreichen wir durch den Zaehler INDREFNR, auf dem gespeichert wird, wie oft Indirekt- Register von anderen Registern mit erinnerungswuerdigem Inhalt durch REFREG referiert werden.

5.4. REGDESCR

```
RECORD
   REGSTAT : ARRAY [ FIRSTREG..LASTREG ] OF REGSTATUS;
   LEVREGS : ARRAY [ LEVRANGE ] OF ACRANGE
END;
```

Die Beschreibung aller Register wird in Records vom Typ REGDESCR zusammengefasst. REGDESCR hat folgende Felder :

a) REGSTAT: Ein Array, das fuer jedes verwaltete Register eine Beschreibung enthaelt.

b) LEVREGS: Dient zum Auffinden von Basisadressregistern ueber den Pegel (siehe FETCH_BASIS).

Vom Typ REGDESCR wurden zwei globale Variablen deklariert: GREGDESCR, die globale Registerbeschreibung und EMPTYREGDESCR, die leere Registerbeschreibung, die zur Initialisierung von GREGDESCR nach Vereinigung von Kontrollfluessen und am Anfang des Prozedurkoerpers dient.

In den Prozeduren, die Kontrollstrukturen compilieren, existieren weitere lokale Variable vom Typ REGDESCR ([3], S. 116 ff.).

6. Die Register-Verwaltungs-Prozeduren

6.1. DECREF (FREG : ACRANGE)

DECREF decrementiert den Referenzzaehler von "FREG". Im Moment des Erloeschens der letzten Referenz (REFNR=0) wird, falls CONTENTS=OTHER ist, CONTENTS auf AVAIL gesetzt, um zu signalisieren, dass das Register keinen erinnerungswuerdigen Wert enthaelt. Ansonsten wird LASTREF auf den Wert von IG gesetzt, um einen LRU-Algorithmus zu ermoeglichen (siehe 6.3). Diese Prozedur ist fuer jedes Register, das angefordert wurde (NEEDREG) und nicht mehr benoetigt wird, aufzurufen.

6.2 DELETEINDIRREFS

Nach Veraenderung der Registerbeschreibung wird DELETEINDIRREFS aufgerufen, um die Beschreibung von indirekt referierten Variablen, die jetzt nicht mehr erinnerungswuerdig sind, zu loeschen. DELETEINDIRREFS wird von NEEDREG, COINZ_DEL und INTERSECTION aufgerufen.

6.3. NEEDREG (VAR VREG : ACRANGE; ANT : ANTTYP)

NEEDREG fordert Register an, wobei die Parameter folgende Bedeutung haben:

ANT kann folgende Werte annehmen :

ONE	bedeutet, dass ein Register benoetigt wird.
TWO.	bedeutet, dass ein Registerpaar benoetigt wird.
PREFFEREDTWO	bedeutet, dass ein Register benoetigt wird, dessen Nachfolger moeglichst nicht belegt ist, da der zu ladende Wert unter Umstaenden einer IDIV-Operation unterworfen wird, die zwei Register benoetigt.
VREG	ist der Ausgabeparameter. Er enthaelt die Nummer des ersten zugewiesenen Registers.

Falls die globale Variable LOADPARAM (s.u.) vom Typ BOOLEAN gesetzt ist und das durch PARREG spezifizierte Register frei ist, wird PARREG zurueckgegeben, anderenfalls wird nach einem LRU-Algorithmus entschieden, welches Register (-paar) zurueckgegeben wird. Die Registerbeschreibung des Registers VREG wird geloescht, CONTENTS auf OTHER und REFNR auf 1 gesetzt.

Falls LOADPARAM auf TRUE gesetzt und das durch PARREG spezifizierte Register belegt ist, wird durch SEARCHCODE, GENERATE_LOAD und EXPRESSION garantiert, dass der Wert des Parameterausdrucks sich auf dem Register "PARREG" befinden wird. Um das plausibel zu machen, betrachten wir die zwei denkbaren Faelle:

i) Auf "PARREG" befindet sich ein Adressausdruck, der fuer das Laden des Parameters benoetigt wird. Dann wird, um den Wert zu laden, die Prozedur GENERATE_LOAD aufgerufen, die als erstes alle Indexregister der zu ladenden Variablen freigibt, also auch PARREG (d.h. dieser Fall kommt nicht vor).

ii) "PARREG" ist durch einen Teilausdruck belegt. Dann ist durch SEARCHCODE sichergestellt, dass bei der spaeteren Verknuepfung der beiden Teilausdruecke das Ergebnis auf das zuerst angeforderte Register (in diesem Fall PARREG) geschrieben wird.

Die einzige Ausnahme stellt die MOD-Operation dar. In diesem Fall wird am Ende von EXPRESSION eine Register-Register-Transfer Operation erzeugt.

NEEDREG ersetzt das in den bisherigen Compiler-Versionen verwendete INCREMENT_REGC.

6.4. INTERSECTION (FREGDESCR : REGDESCR);

INTERSECTION bestimmt den Mengendurchschnitt von FREGDESCR und GREGDESCR . Diese Prozedur wird nach Vereinigung von Kontrollfluessen (IF-THEN-ELSE-Anweisungen) aufgerufen, um die gueltige Registerbeschreibung auf GREGDESCR zu erzeugen.

6.5. FUNCTION SEARCH (FATTR : ATTR) : ACRANGE;

SEARCH durchsucht die globale Registerbeschreibung nach dem durch FATTR spezifizierten Datum. Falls dieses Datum gefunden wird, wird die entsprechende Registernummer zurueckgegeben, anderenfalls wird 0 zurueckgegeben.

6.6. COINZ_DEL(FATTR : ATTR; FREG : ACRANGE)

COINZ_DEL loescht alle Registerbeschreibungen, die nicht mehr gueltig sein koennten, nachdem der in FREG enthaltene Wert auf eine Variable (beschrieben durch FATTR) weggespeichert wurde ([3],S.29ff).

6.7. ENTERPARAM (FCP : CTP; FREG : ACRANGE; FLENG : LENGTHTYP)

Da am Anfang eines ProzedurKoerpers der Registerzustand durch die Parameteruebergabe definiert ist, haben wir dieses ausgenutzt, indem wir die globale Registerbeschreibung entsprechend der Parameterliste initialisiert haben.

6.8. ENTERREGDESCR (FREG : ACRANGE; FATTR : ATTR)

Die Beschreibung des durch FATTR spezifizierten Datums wird auf GREGDESCR.REGSTAT [FREG] eingetragen.

7. Notwendige Aenderungen im Compiler

Jede codeerzeugende Prozedur musste geaendert werden. Bei den folgenden Prozeduren waren umfangreiche Aenderungen notwendig:

7.1. STORE (FREG : ACRANGE; FATTR : ATTR)

Mit Hilfe von COINZ_DEL werden moegliche Koinzidenzen zwischen Registerbeschreibungen und der Variablen, die durch FATTR beschrieben wird, erkannt und entsprechend behandelt. Falls der abzuspeichernde Ausdruck (dessen Wert auf FREG steht) nicht erinnerungswuerdig ist, wird die Beschreibung der Variablen,

auf die abgespeichert werden soll, auf GREGDESCR.REGSTAT[FREG] eingetragen. Andernfalls wird die Registerbeschreibung nicht veraendert.

Bei der Compilation von I := 3 steht die Beschreibung des betreffenden Registers auf (CNST,3), waehrend bei I := I+2 die Beschreibung von I eingetragen wird.

7.2. Compilerprozedur SEARCHCODE

In dieser Implementationsstufe sind wir davon ausgegangen, dass auf Registerinhalte, auf denen arithmetische Operationen stattfinden koennen, nicht mehr als eine Referenz besteht. Der SEARCHCODE zugrundliegende Algorithmus [1] wurde von uns in folgender Weise modifiziert:

```
falls noch kein Teilbaum geladen wurde,
        dann  falls einer der beiden Teilbaeume in der
                    Registerbeschreibung vorhanden ist
                und nicht referiert wird
                    oder die Operation ist nicht zerstoerend
                dann wird eine Referenz auf ihn erzeugt
              sonst falls der rechte Teilbaum gepackt ist
                     dann lade ihn
                    sonst lade den linken Teilbaum
      (* Zusicherung : mindestens ein Teilbaum ist geladen *)

falls der linke Teilbaum geladen ist,
dann erzeuge Code
sonst (* linker Teilbaum nicht geladen *)
      aendere Instruktion (z.B. < => > ) und erzeuge Code

falls  eine zerstoerende Operation stattfand,
dann   aendere Registerbeschreibung
```

7.3. Compilerprozedur FETCH_BASIS

Durch die Erinnerungswuerdigkeit von Basisadressen beginnt jetzt die Bestimmung von Basisadressen fuer einen Pegel P bei demjenigen Basisregister i des Pegels Q[i] fuer das gilt : Fuer alle j mit Q[j] >= P folgt Q[j]-P >= Q[i]-P . Dadurch reduziert sich die Code-Erzeugung bei Zugriffen auf Variable, die sich weder auf Hauptprogramm- noch auf aktuellem Niveau befinden.

7.4. GENERATE_LOAD (FINSTR : INSTRANGE; VAR FATTR : ATTR; IDIV : BOOLEAN)

GENERATE_LOAD behandelt alle Speicher-Register-Transfer-Instruktionen, wie z.B. MOVE, DMOVE, FLTR, MOVM. Im alten Compiler wurden diese Instruktionen zusammen mit arithmetischen und logischen Instruktionen behandelt. Da die Struktur dieser beiden Operationsklassen

stark unterschiedlich ist, haben wir die Behandlung der Speicher-Register-Transfer-Befehle aus GENERATE_CODE ausgelagert.

Da IDIV-Operationen zwei aufeinanderfolgende Register benoetigen, wird durch den Parameter IDIV angezeigt, ob der zu ladende Wert einer solchen Operation unterzogen werden soll. Falls IDIV gesetzt ist, sorgt GENERATE_LOAD dafuer, dass das Datum auf ein Register transferiert wird, dessen Nachfolger nicht belegt ist. Die Registerbeschreibung wird gemaess FATTR aktualisiert, FATTR.KIND auf EXPRESSION gesetzt und FATTR.REG eingetragen.

7.5. CALL- und CALL_NON_STANDARD

Die Parameter in Prozeduraufrufen werden weiterhin in den Registern aufsteigend von Register 2 ab uebergeben. Um zu erzwingen, dass der aktuelle Parameter in das entsprechende Register geladen wird, haben wir die Variablen LOADPARAM und PARREG eingefuehrt. Falls LOADPARAM nicht gesetzt ist, geschieht die RegisterallokAtion normal, ansonsten wird, soweit moeglich, das durch PARREG spezifizierte Register alloziert. Da es moeglich ist, dass innerhalb von Parameterlisten weitere Parameterlisten auftauchen, muessen PARREG und LOADPARAM innerhalb von CALL rekursiv verwaltet werden. Nachdem die Parameterliste abgearbeitet worden ist, wird bei Nicht-Standard-Prozeduren die globale Registerbeschreibung geloescht. Bei Standard-Prozeduren wird abhaengig von der aktuellen Prozedur die Registerbeschreibung geloescht oder aktualisiert. Bei Nicht-Standard-Funktionsaufrufen werden die referierten Register in den LaufzeitKeller gerettet und nach dem Aufruf zurueckgeladen. Die Registerbeschreibung wird dem aktuellen Stand angepasst (Prozeduren SAVEREFREGS und RELOADREFREGS). Die StandardfunKtionen werden wie die Standardprozeduren behandelt.

7.6 Compilerprozedur IFSTATEMENT

Abweichend von der Ammann-Implementation wird nach der Compilation einer IF-THEN-ELSE-StruKtur die globale Registerbeschreibung nicht geloescht, sondern es wird der Mengendurchschnitt der Registerbeschreibungen bei Vereinigung der Kontrollfluesse gebildet.

7.7. KontrollstruKturen

Die Compilation der KontrollstruKturen ist der neuen Registerverwaltung angepasst.([3] s.116 ff)

7.8. Compilerprozedur WITHSTATEMENT

Bei der alten Implementation wurden die von With-Anweisungen erzeugten anonymen Variablen permanent in den Registern

behalten. Bei der jetzigen Implementation ist diese Strategie weder sinnvoll noch einfach zu verwirklichen, da es zu Blockierungen bei der Parameteruebergabe kommen koennte. Die anonyme Variable wird deshalb im Datensegment der aktuellen Prozedur aufbewahrt, befindet sich aber mit grosser Wahrscheinlichkeit zusaetzlich in einem Register. Die Vorteile dieser Strategie gegenueber der alten sind:

a) Ein Retten der With-Register vor Prozeduraufrufen entfaellt.
b) Zur Auswertung von Ausdruecken innerhalb von With-Anweisungen stehen alle verwalteten Register zur Verfuegung.
c) Der Transfer der anonymen Variablen in den With-Register-Keller entfaellt.

7.9 Compilerprozedur SELECTOR

Die Code-Erzeugung bei Array-Zugriffen kann effektiver gestaltet werden. Das Abziehen der unteren Array-Grenze (LMIN) kann in den Faellen unterlassen werden, in denen ein arithmetischer Ueberlauf nicht zu befuerchten ist. Bei Zugriffen auf gepackte Array-Komponenten braucht nicht die Adresse des Bytepointers explizit berechnet zu werden. Dazu muss in den Record ATTR ein neues Feld BPINDEXR eingefuehrt werden, das das Indexregister bezueglich der Bytepointeradresse enthaelt.

7.10. Zusaetzliche Aenderungen im ATTR-Record

Bei der alten Implementation wurde innerhalb von FETCHBASIS das Feld VLEVEL auf 1 gesetzt, um zu verhindern, dass FETCHBASIS mehrere Male aktiviert wird. Da durch diesen Eingriff die Information ueber den Pegel der Variablen verloren geht, wir diese Information aber fuer das Nachtragen der Registerbeschreibung benoetigen, wurde ein neues Feld FBDONE vom Typ BOOLEAN in ATTR eingefuehrt. Um moegliche Koinzidenzen zwischen Variablen zu erkennen, benoetigen wir ausserdem Informationen ueber die Zugriffsart, die von SELECTOR in das neue Feld ACCESS von ATTR eingetragen werden.

8. Sonstige Aenderungen

Waehrend der Arbeit am Compiler haben wir zusaetzlich diverse Fehler beseitigt, Codeoptimierungen implementiert und die Codeerzeugung fuer Laufzeittests erweitert.

8.1. Beseitigte Fehler

8.1.1. TRUNC war falsch implementiert. Die Funktion lieferte den bei negativen Argumenten naechst kleineren ganzzahligen Wert.

8.1.2. Die Funktionen MIN und MAX waren fehlerhaft, der Aufruf konnte zu Endlosschleifen oder falschen Ergebnissen fuehren.

8.1.3. Der Aufruf von MESSAGE fuehrte zu Laufzeitfehlern, wenn als letztes Argument ein SET oder ein vom Benutzer deklarierter Skalar angegeben war.

8.1.4. Bei der Parametersubstitution von formalen Prozeduren oder Funktionen konnte es vorkommen, dass eine falsche Startadresse uebergeben wurde.

8.1.5. Der TTY-Eingabefile wurde nicht eroeffnet, wenn dieser lediglich als aktueller Parameter bei Prozedur- oder Funktionsaufrufen auftrat.

8.1.6. Wenn bei Aufruf der Prozedur NEW ein Argument angegeben wurde, das die Groesse eines dynamisch zu erzeugenden Arrays bestimmt, so wurde im Widerspruch zur Definition ein Array mit "Argument - untere Grenze des Arrays (LMIN) + 1" Komponenten erzeugt.

8.1.7. Wenn die Programmparameter mittels Kommando (CMD) File besetzt wurden und gleichzeitig das Debug-System aktiv war, kam es zu einer Endlosschleife.

8.2. Codeoptimierungen

8.2.1. Wir haben die KI-10 Instruktionen FLTR, FIX, FIXR, DMOVE und DMOVEM in die Codeerzeugung integriert. Insbesondere sei eine Stelle in der Compilerprozedur ENTERBODY erwaehnt: Parameter werden in den Registern uebergeben und muessen deshalb beim Prozedureintritt in das lokale Datensegment kopiert werden. Da die Parameter zumeist aufeinanderfolgende Speicherstellen belegen, konnten wir auch an dieser Stelle die Instruktion DMOVEM verwenden, und so eine Instruktion sparen.

8.2.2. Bei Zugriffen auf ungepackte Arraykomponenten kann das Abziehen der unteren Arraygrenze (LMIN) waehrend der Laufzeit in den Faellen unterlassen werden, in denen ein arithmetischer Ueberlauf bei der Berechnung des Wertes "Indexausdruck * Komponentengroesse" nicht zu befuerchten ist. Der Wert "untere Arraygrenze * Komponentengroesse" kann dann waehrend der Compilationszeit von der Relativadresse (DPLMT) abgezogen werden. Bei dieser Art des Zugriffes spart man bei mehrstufigem indizierten Zugriff je eine Instruktion.

8.2.3. Bei Zugriffen auf gepackte Arraykomponenten muss die Bytepointeradresse nicht explizit berechnet werden, so dass auch in diesem Fall eine Instruktion gespart werden konnte.

8.2.4. Wenn auf das erste Feld eines Record zugegriffen wird, dieses ungepackt ist und wenn das indirekt Bit gesetzt ist, so braucht nicht die Adresse des Feldes geladen werden, da man ja die Indirektadressierung ausnutzen kann.

8.3. Zusaetzliche Laufzeittests

Nach Vorbild des CDC-6000-Compilers haben wir zwei weitere Moeglichkeiten fuer die Erzeugung von Laufzeittestcode implementiert.

8.3.1. Bei Zugriffen mit Hilfe von Zeigern wird geprueft, ob der Zeiger in die Halde zeigt.

8.3.2. Bei der Parametersubstitution von formalen Wertepparametern, deren Typ ein Unterbereich ist, wird geprueft, ob der Wert des aktuellen Parameters innerhalb der Grenzen des Wertebereichs liegt.

9. Messresultate

9.1. Speicherbedarf fuer eine Selbstcompilation

Der Programmcode (High-Segment) ist nicht laenger geworden, d.h. der Codeanteil, der durch die Optimierung weggefallen ist, ist durch die zusaetzlichen Prozeduren wieder dazugekommen. Allerdings benoetigt der Compiler fuer eine Selbstcompilation rund 3K Worte mehr im Datensegment (Low-Segment), da die Registerbeschreibungen, die ja ziemlich gross sind, rekursiv fuer jede Kontrollstruktur gespeichert werden muessen.

9.2. Codereduktion fuer Benutzerprogramme

Wir haben die im Kapitel 3 erwaehnten Programme bzw. Programmteile fuer die Messung benutzt. Bei den folgenden Werten steht (a) fuer vom alten Compiler erzeugte Codelaenge und (n) fuer die vom neuen Compiler erzeugte Codelaenge.

9.2.1. Program EASTERN
(a) 135 Worte, (n) 124 Worte,
relativer Gewinn: 8.1 %

9.2.2. Compilerprozedur ENTERID
(a) 115 Worte, (n) 77 Worte,
relativer Gewinn: 23.5 %

9.2.3. Compilerprozedur COMPTYPES
(a) 254 Worte, (n) 224 Worte,
relativer Gewinn: 11.8 %

9.2.4. Der neue Compiler selber
(a) 27K + 771 Worte, (n) 25K + 190 Worte,
relativer Gewinn: 9.3%

9.3. Verlaengerung der Compilationszeit

Bei der Selbstcompilation stellt sich heraus, dass der alte Compiler (1.55 min) rund 17% schneller ist als der neue Compiler (2.15 min).

10. Weitere Ideen zur Compileroptimierung

10.1. Innerhalb der Registerverwaltung

10.1.1. Eine Umorganisation der Felder von REGDESCR durch Einfuehren weiterer TAG-Felder und/oder eventuell durch Packen von REGDESCR Koennte sich die Groesse des LaufzeitKellers, den der Compiler benoetigt, reduzieren lassen (Durch die Registerverwaltung vergroesserte sich der LaufzeitKeller um ca. 3K).

10.1.2. Man Koennte auch bei arithmetischen Operationen erlauben, dass die REFNR des betreffenden Registers groesser als Eins wird. Der Gewinn dieser Massnahme ist allerdings fraglich, da die Verwaltung in diesem Fall erheblich Komplizierter werden wuerde.

10.1.3. Es waere zu ueberpruefen, wie gross die Codeersparnis durch die Prozedur INTERSECTION ist, um diese Prozedur gef. aus dem Compiler zu entfernen. Da INTERSECTION eine laufzeitintensive Prozedur ist, Koennte man eine ReduKtion der Laufzeit des Compilers erreichen.

10.1.4. Register 0 wird zur Zeit nicht von der Registerverwaltung benutzt. Wenn man in Register 0 permanent den Wert 1 halten wuerde, waeren Zuweisungen von 1 bzw Vergleiche mit 1 erheblich effeKtiver zu compilieren. In diesem Fall waeren zusaetzlich Aenderungen in der Laufzeitunterstuetzung (LZU) erforderlich, da die LZU Register 0 benutzt.

10.1.5. In der bestehenden Implementation unterscheidet die Registerverwaltung nicht zwischen durch Zeiger referierten Variablen und Variablenparametern. Beide sind durch ihre indireKte Zugriffsart gekennzeichnet. Das hat zur Folge, dass bei Abspeicherungsoperationen diese beiden Variablenarten gleich behandelt werden, obwohl Koinzidenzen zwischen ihnen ausgeschlossen sind. Man sollte daher in ACCESSKIND eine zusaetzliche Unterscheidung einfuehren.

10.1.6. Bei der Compilation von Parameterlisten (LOADPARAM = TRUE) erzeugt die Registerverwaltung Keine gestreute Registerbelegung, sondern arbeitet praKtisch nur noch auf dem durch PARREG spezifizierten Register, was einer reichhaltigen Registerbelegung entgegenwirKt. Man sollte daher in codegenerierenden Prozeduren, von denen bekannt ist, dass sie nicht die letzte InstruKtion bei der Compilation des aKtuellen Parameters erzeugen, LOADPARAM lokal zurueckSetzen, um eine gestreute Registerbelegung zu erreichen. In Frage Kommende Prozeduren waeren INDEXCODE und FETCHBASIS.

10.2. Unabhaengis von der Registerverwaltung

10.2.1. Falls der Wert von NIL von 377777B auf 0B abgeaendert werden wuerde, waeren Zuweisungen von NIL bzw Vergleiche mit NIL effektiver zu compilieren. Diese Anderung waere ohne grosse Probleme durchzufuehren, da wir einen Laufzeitzeigertest (Siehe 8.3.1) implementiert haben.

11. Ablauf der Arbeit

11.1. Geplante Zeiteinteilung

Fuer das gesamte Projekt standen uns 2 Monate zur Verfuegung. Das erste Drittel dieser Zeit war fuer das Einarbeiten in die Literatur einschliesslich Compilerlisten vorgesehen. In der verbleibenden Zeit wollten wir uns der Implementation und Dokumentation widmen.

11.2. Der tatsaechliche Zeitplan

21.6.77 Erste Besprechung mit Herrn Nagel und Aufstellung der Ziele:

1. Beseitigung von Fehlern
2. Implementation einer neuen Registerverwaltung
3. Boolesche Operatoren durch Sprungketten uebersetzen
4. KI-10 Instruktionen einbauen
5. Umstellung der TTY-Unterstuetzung, wobei wir uns auf die Punkte 2. und 3. Konzentrieren wollten.

15.8.77 Beginn des Projekts

15.8-7.9.77 Einarbeitung in die Literatur

7.9.77 Vorlage der ersten schriftlichen Konzeption der Registerverwaltung sowie Erorterung der damit verbundenen Probleme (Abschnitt 1-4 und Teile von 5 und 6).

8.9.-30.9.77 Einbau der KI-10 Instruktionen, Beseitigung von Fehlern und Codierung der Registerverwaltungsprozeduren.

1.10-7.10.77 Teilimplementation der Registerverwaltung.

7.10.77 Vorlage des vorlaeufigen Abschlussberichts (Kapitel 1-5 und grosse Teile von 6)

8.10.-13.10.77 Vollstaendige Implementation der neuen Registerverwaltung. Leider sind wir in Folge der Zeitknappheit nicht mehr zum Validieren gekommen.

Um den Compiler abschliessend zu testen, wurden uns in den Weihnachtsferien noch zwei zusaetzliche Wochen zu Verfuegung gestellt.

17.12.-24.12.77 Austesten des Compilers, weitere Fehlerbeseitigung.

24.12.77 Erste geglueckte Selbstcompilation.

25.12.-30.12.77 Austesten des Compilers mit Debug und der PASCAL-LZU.

Nach einem weiteren 1/4 Jahr kamen wir dann endlich dazu, diesen Bericht fertigzustellen.

Literaturverzeichnis

[1] Grosse-Lindemann, C.-O. Weiterfuehrende Arbeiten am PASCAL-Compiler Diplom-Arbeit am IFI-Hamburg (1974)

[2] Ammann, U Die Entwicklung eines PASCAL-Compiler nach der Methode des strukturierten Programmierens Dissertation ETH Zuerich (1975)

[3] Ammann,U On Code-Generation in a PASCAL-Compiler Bericht Nr. 13 des IFI-Zuerich (1976)

[4] Kisicki, E., und Nagel, H.-H. PASCAL for the DECSystem-10 Mitteilung Nr. 37 des IFI-Hamburg (1976)

[5] Jensen, K und Wirth, N PASCAL User Manual and Report Lecture Notes in Computer Science, Nr. 18 Springer-Verlag, New York (1976)

[6] DECSystem10-PASCAL-Compiler Liste Version vom 30-DEC-76

[7] CDC 6000-3.4-PASCAL-Compiler-Liste Version vom Maerz 1976

Teil 2

Wartungsarbeiten am DECsystem-10 PASCAL Compiler
ausgehend von Version vom 30.12.76 (PASRLX.PAS)
und Version vom 17.12.77 (PASRLZ.PAS)

I. Fehler

1. Die Funktionen MIN und Max sind fehlerhaft wenn
   a) mehr als zwei Argumente angegeben werden. Führt zu einer Totschleife.
   b) sie verschachtelt aufgerufen werden, z.B. min(i, max(j,k)); führt zu inkorrekten Resultaten.
   Beseitigt durch Brügge, Mühle, Nebel

2. Die Funktion TRUNC (Pascal User Manual and Report S. 107, S. 161). Das Resultat bei Anwendung auf negative Integer-Zahlen liefert eine negative Integer-Zahl, deren absoluter Betrag kleiner ist als der des Argumentes.
   Beseitigt durch Nebel

3. Falls die aktuellen Programmparameter mit Hilfe eines "CMD"-Files übergeben werden, gerät das Debug-System nach Ausgabe von "$" in eine Totschleife.
   Beseitigt durch Mühle, Nebel

4. Bei der Ausführung einer RESET-Anweisung ist die PPN bei darauf folgendem GETSTATUS nicht mehr vorhanden. Ursache: Das LOOKUP-UUO überschreibt in der LZU-Routine RESET den Platz der PPN mit negativem Blockcount.
   Beseitigt durch Mühle

   Der Fileblock wurde um die vier Worte
   FILKNM
   FILKEX
   FILKPR
   FILKPP
   erweitert. RESET kopiert vor einem LOOKUP-UUO die aktuellen Parameter in diese Plätze, so daß die PPN erhalten bleibt und gleichzeitig der von dem LOOKUP-UUO zurückgegebene Blockcount verfügbar ist.

5. Der Versuch, eine Variable eines Subrange-Typen, der genau 35 Bit erfordert, zu packen, führt in eine Totschleife.
Beseitigt durch Brügge

6. Bei manchen Laufzeitfehlern wird vom Debug-System eine falsche Zeilennummer angegeben, i.a. die letzte des Programms.
Beseitigt durch Mühle

7. Falls das Debug-System nur lokal ausgeschaltet wird, gibt es einen Ill. Mem. Ref.
Beseitigt durch Mühle, Nebel

8. Falls das TTY schon eröffnet ist, aber das Device ungleich 'TTY...' gewesen ist, muß das TTY nochmals initialisiert werden (bei neuem Compiler wegen Wegfalls von TTY gegenstandslos geworden).
Beseitigt durch Brügge

9. Vergleiche von Elementen PACKED ARRAY[ ]OF CHAR: Das Linkeste Bit wird als Vorzeichen-Bit interpretiert, obwohl es Darstellungs-Bit ist. Vergleiche: IfI-HH-M-37/76, S. 52 Punkt 9.2 a+b
Beseitigt durch Mühle

10. Die IN-Prüfung bei SETs prüft nicht, ob die Maximalzahl von Elementen im SET überschritten wird.
Beseitigt durch Brügge

11. Wenn der Wert 377777(=NIL) in einem Indexregister steht, tritt unter Umständen kein Ill. Mem. Ref. auf, sofern gelesen wird, z.B. aus dem Highsegment. Zumindestens bei der Option T+ (durch andere Implementation von Zeigervariablen sowie durch Einführung von Bereichsprüfungen gegenstandslos geworden).
Beseitigt durch Mühle, Nebel

12. Das TTY wird dann nicht eröffnet, wenn es nur in Standardprozeduren, z.B. GETFILENAME, als Argument auftritt. Ursache: Das Setzen der Schaltervariablen TTYREAD im Compiler bedeutet, daß eine Eingabe vom TTY erfolgen wird. Argumente von Standardprozeduren werden dafür jedoch nicht untersucht (inzwischen gegenstandslos geworden).

13. Wird die NEW-Prozedur für einen Record ohne Tagfieldidentifier mit einem Parameter für den Tagfieldidentifier aufgerufen, so gibt es einen Ill. Mem. Ref. Ursache: Der Compiler generiert Code zur Initialisierung des nicht existierenden Tagfields.
Beseitigt durch Brügge

14. Die Initialisierung von gepackten Records in INITPROCEDURE wird vom Compiler nicht als Fehler erkannt. Ursache: Fehlende Abfrage.
Beseitigt durch Brügge

15. Zwei Subrange-Typen, von denen der eine genau ein Wort belegt, die gepackt werden sollen, werden irrtümlich in ein Wort gepackt.
Beseitigt durch Brügge

16. Mitunter erscheint irrtümlich die Fehlermeldung des Debug-Systems $ STOPTABLE DESTROYED. Ursache: Fehlende Initialisierung von GPAGE in der Prozedur LINEINTERVAL.
Beseitigt durch Brügge

17. Der Compiler erlaubt die Verwendung von Subrange-Typen von REAL als Array-Indizes, was nach dem Report nicht erlaubt ist.
Beseitigt durch Brügge

18. ORD("REALCONST") führt zu falschen Werten.
Beseitigt durch Mühle

19. a) Wird bei der Abfrage nach der Ausgabedatei als Dateiname lediglich TTY:, also nur der DEVICE-Name, angegeben, geht die gesamte Ausgabe auf DSK:OUTFIL.E. Ursache: Es muß geprüft werden, ob nur FILENAME ≠ '____' abgefragt wird, und nicht auch DEVICE,PROTECTION etc.
Beseitigt durch Mühle

b) Wird bei der Abfrage nach der Ausgabedatei TTY:"DATEINAME" angegeben, gerät das übersetzte Programm in eine Totschleife.
Wird dasselbe Programm mit D+ übersetzt und TTY:"DATEINAME" als Ausgabedatei angegeben, gerät das Debug-System in eine

Schleife, in der endlos vom TTY eingelesen wird.
(Entfällt durch andere Implementation der Terminalbedienung)

20. Ein Programm, das im Gesprächsbetrieb lief, stieg im Stapelbetrieb aus mit %? INPUT ERROR: RESET REQUIRED FOR TTY.
(Erledigte sich durch Beseitigung eines anderen Fehlers.)

21. Die Case-Anweisung funktioniert bei betragsmäßig sehr großen Case-Marken nicht.
Beseitigt durch Brügge

22. Die Standardprozedur MESSAGE führt zu einem ILL. UUO, wenn man als Argument den Wert eines skalaren Typen angibt.
Beseitigt durch Mühle, Nebel

23. Bei großen Programmen mit ca. 40 - 50 SOS-Seiten gibt das Debugsystem bei Post-Mortem-Dumps (nur dort?) in PROCEDURE BACKTRACING anstelle der Zeilennummern nur xxxxx aus. Mögliche Ursache (Kisicki): In Prozedur NEWPAGER (o.ä., auf jeden Fall in der Nähe von MACRO1) sind die Parameter RHALF, LHALF in der Reihenfolge vertauscht.
Beseitigt durch Kisicki

II. Optimierungen

1. Entfernung der Routinen INTREA, ROUND und TRUNC aus LZU, da die Aufgaben inline erledigt werden.
(Mühle, Nebel)

2. In MAIN des Compilers wird noch OPTION aufgerufen. Ist überflüssig (führte zum Laden der CCL-Routinen mit 1.5 kWorten).
(Mühle, Nebel)

3. Untersuchung, ob die Umstellung INPUT statt TTY und OUTPUT statt TTY sinnvoll ist, d.h. Ausgabe auf Dateien findet nur noch statt, wenn der Filebezeichner explizit angegeben ist. Vorteil: Die Fehlererholung nach Konversionsfehlern bei falscher Eingabe wäre leichter realisierbar. Inzwischen durchgeführt.
(Brügge)

4. NIL=377777 ersetzen durch NIL = Ø und optimieren.
   (Mühle)
5. Der neue Compiler erzeugt eine um 2 Instruktionen kürzere Befehlsfolge für den AND-Operator. (Mühle)

III. Benutzerfreundlichkeit

1. Bei Konversionsfehlern in einer Texteingabe über das TTY automatische Fehlernachricht und Wiederholung der Eingabe.
   (Mühle)

2. Änderung des E/A-Modus von ASCII auf ASCIL, damit die Eingabe auch mit Altmode beendet werden kann.
   (Mühle)

3. Hinter dem Gleichheitszeichen sollte im Debugsystem ein Leerzeichen folgen: name= value.
   (Mühle)

4. Ausgabe einer Warnung bei bestimmten Fehlern

   Die Prozeduren ENDOFLINE und ERROR wurden von Herrn Brügge so modifiziert, daß bei "ERROR IN OPTION" die Variable ERRORFLAG nicht gesetzt und nur eine Warnung auf das Terminal geschrieben wird.

5. Option zur Ausgabe der Maschineninstruktionenfolge

   wird bei Fehlern nicht mehr abgeschaltet.

6. Ausgabe aller aktiven Optionen

   Implementiert (bisher nur für Ausgabe auf Terminal). Muß noch erledigt werden für Ausgabe auf LIST-File (sofern ein solcher erzeugt wird!)

7. Fehlermeldung bei Indexbereich > 2 **17 - 1

   Implementiert

IV. Änderungen der vom Compiler akzeptierten Sprache

Aufgrund internationaler Vereinbarungen zur Vereinheitlichung gewisser Spracherweiterungen wurde die folgende Änderung vorgenommen:

Die Erweiterung der CASE-Anweisung wurde von

OTHERS : <statement> END auf

OTHERWISE : <statement> {; <statement>} END

umgestellt. Im Gegensatz zu vorher können nunmehr beliebig viele Anweisungen zwischen OTHERWISE: und dem die CASE-Anweisung abschließenden END auftreten.
(Faasch)

V. Verschiedenes

(K. Mühle)

Problem

In DEBUG ist es nicht mehr möglich, eine vom Benutzerprogramm noch nicht verarbeitete TTY-Eingabe an das Benutzerprogramm zurückzugeben, da der entsprechende Puffer im Monitor sitzt. Ein entsprechender Hinweis (INPUT DELETED: ...) fehlt zur Zeit.
Zusätzlich ist noch das ControlZ so implementiert worden, daß es EOF (TTY) auf TRUE setzt (DECSystem-10 Assembly Language Handbook, Seite 5 - 29).

Programmkopf

Bei Angabe von Filebezeichnern im Programmkopf werden zur Laufzeit nur die Datei-Spezifikationen abgefragt. D.h. alle Dateien (außer INPUT und OUTPUT) müssen vom Benutzer durch RESET bzw. REWRITE eröffnet werden. Die aktuellen Spezifikationen werden im Fileblock übergeben (Prozeduren READPROGRAM_PARAMETER und SETSTATUS). Da auch INPUT und OUTPUT als Programm-Parameter erscheinen können, ist es nötig, die I/O-Routinen so zu programmieren, daß sie unabhängig von der im Fileblock enthaltenen Dateispezifikation funktionieren. Dies

führte zur Erweiterung des Fileblocks um das Wort FILTTY. Dabei bedeutet

FILTTY = +1 die Datei ist dem Gerät TTY zugeordnet
FILTTY = -1 die Datei ist einem TEMPCOR-File zugeordnet
FILTTY = 0 sonst

Der Compiler erhielt den Programmkopf

PROGRAM PASCAL(OBJECT, LIST, SOURCE)

und die Prozedur GETFILENAME wurde aus dem Compiler entfernt. Ersparnis: ca. 300 Zeilen Quellcode.

Zahlenmaterial

Der neue Compiler erzeugt bei einer Selbstcompilation mit den Optionen /NOCHECK/NOLIST ca. 23K500 Worte im Highsegment. Eine Compilation mit dem SYS-Compiler erzeugt ca. 26K1000 Worte, d.h. der von dem neuen Compiler erzeugte Code ist um 3600 Worte bzw. 15 % dichter.

Der Anteil der einzelnen Optimierungen ist dabei wie folgt:

Registerallokation	2242 Worte
AND-Optimierung:	320 Worte
NIL-Optimierung:	350 Worte
KI-10 Instruktionen:	701 Worte
	3613 Worte

Der neue Compiler (12600 Zeilen) benötigt für eine Selbstcompilation ca. 2 min 20 sec CPU-Zeit. Das entspricht einer Geschwindigkeit von 92 Zeilen/sec. Er ist damit geringfügig langsamer als der SYS-Compiler (96 Zeilen/sec), d.h. der Zeitverbrauch für die Optimierung der Registerverwendung während der Compilation ist etwas größer als der Zeitgewinn durch den optimierten Objektcode des Compilers. Bei allen anderen Programmen - außer dem Compiler selber - ist ein echter Laufzeitgewinn realisiert.