DAS PEARL-KOMPILIERSYSTEM FÜR DIE SIEMENS 300-16 Bit

Dipl.-Ing. K.F. Bamberger
Fa. Siemens AG - E STE 3 - 7500 Karlsruhe

Als sich im Jahre 1969 Vertreter von Herstellerfirmen, Instituten und Softwarehäusern trafen, um die Arbeit an einer Prozeßsprache aufzunehmen, war in unserem Hause die Forderung der Programmierer nach einer solchen Sprache sehr stark.

Die Prozeßrechner-Programmierer benutzten - mit wenigen Ausnahmen - Assemblersprache (Ausnahmen waren z.B. Prozeßmodelle oder Programme für Lastflußrechnungen, die in ALGOL 60 geschrieben wurden) und waren unzufrieden über das Ausprogrammieren immerwiederkehrender Routinen, die Fehlerquote bei der Notierung, die schwere Verständlichkeit der Protokolle und die Mühe, die es machte, sich in Programme anderer hineinzufinden, diese zu ändern bzw. zu ergänzen.

Wir hatten deshalb bereits begonnen, Unterprogramm- oder Makrobibliotheken aufzubauen - Makroassembler waren die logische Konsequenz - wir arbeiteten eine Assemblernotation aus, die insbesondere für Adreßrechnung und arithm. Verknüpfungen an Höhere Sprachen (FORTRAN) heranreichte und heranreicht, FORTRAN wurde um Prozeß-Calls erweitert, zusätzlich entstanden die technol. orientierten Standardsysteme (MADAM, SOSYNAUT, CEPAMAT, SIMAT und eine Reihe anderer).

Dieser sehr kurze Rückblick sollte einerseits zeigen, daß die Suche nach leistungsfähigen Programmerstellungsmitteln und damit auch nach einer Prozeßsprache bereits damals im vollen Gange war, andererseits aber auch die Umgebung umreißen, in die die Sprache PEARL, zumindest bei uns, hineingestellt werden mußte und in der sie sich auch heute noch behaupten muß.

Die diesem Bericht zugrunde liegenden Arbeiten wurden mit Mitteln des Bundesministers für Forschung und Technologie gefördert. Die Verantwortung für den Inhalt liegt jedoch allein beim Autor.

Siemens hat also von Anfang an aktiv an der PEARL-Entwicklung mitgearbeitet und 1976 einen Pilotcompiler fertiggestellt, der sich an den bis dahin veröffentlichten PEARL-Sprachmitteln orientierte. Diese Pilotentwicklung machte es uns dann auch möglich, bereits im Januar 1978 einen Basis-PEARL-Compiler für den Prozeßrechner PR 330 freizugeben, obwohl erst im September 77 der einzig verfügbare, allgemein anerkannte Subset von Basis-PEARL verabschiedet worden war bzw. erschienen war. (Bild 1).

Das Siemens-PEARL-Kompiliersystem

Zunächst möchte ich einige Entwurfs- und Realisierungskriterien des PEARL-Kompiliersystems, die die Struktur des Compilers wesentlich beeinflußt haben, auflisten.

1. Integration des PEARL-Compilers in die vorhandene Systemprogramm-Umgebung:
   - Ablauf des PC unter den Standard-Organisationsprogrammen der Prozeßrechnersysteme 300-16 Bit
   - Einhalten der für Systemprogramme geltenden Bediensyntax
   - Verwenden vorhandener Dienst- und Hilfsprogramme wie Binder, Lader, Monitor
   - Archivierung, Compilerproduktion und Fehlerdiagnose mit bewährten Verfahren

Damit ist auch zum Ausdruck gebracht, daß es nicht genügt, nur einen Compiler zu realisieren, sondern daß das Vorhandensein einer entsprechend leistungsfähigen Systemprogrammumgebung mindestens genau so wichtig ist.

2. Ein PEARL-Programm muß sich wie jedes andere Anwenderprogramm verhalten:
   - Ablauf unter Standard-Organisationsprogrammen
   - Zusammenwirken mit Nicht-PEARL Programmen oder Programmpaketen

3. Keine Unterscheidung zwischen Übersetzungs- und Zielrechner, d.h. eine Kompilierung muß sowohl im Rechenzentrum als auch vorort on-line möglich sein:
   - Einhaltung eines vorgegebenen Laufbereiches
   - Geringe Transferbelastung zum Peripherspeicher (solche Transfers bilden i.a. einen Engpaß bei Prozeßaufgaben)

4. Wirkungsvolle Unterstützung beim Testen von PEARL-Programmen:
   - Leistungsfähige Testhilfen zur Laufzeit
   - Prägnante Fehlertexte im Kompilierprotokoll am Ort des Fehlers
   - Hohe Kompiliergeschwindigkeit

   Die Bilder 2 u. 3 sollen diese Aussagen noch etwas mehr veranschaulichen.

Die Forderung nach einem portablen Compiler hatten wir nicht und zwar, weil wir aus den Untersuchungen am Pilotcompiler sicher zu sein glaubten, Forderungen wie:

> Schnelle Übersetzungszeiten und Compiler als Systemprogramm (z.B. Laufbereich)

mit einem solchen portablen Compiler nicht zusätzlich erfüllen zu können.

Bild 4 soll schematisch aufzeigen, mit welchen Mitteln erreicht wurde, diese z.T. widersprüchlichen Forderungen einer befriedigenden Lösung zuzuführen.

Die naheliegende Lösung ist ein Mehrpaßcomiler, wobei umfangreiche logisch einheitliche Aufgaben und mehrere Pässe (z.B. Syntax-Analyse auf Pässe 2, 3 und 5) verteilt wurden.

Ein solcher Mehrpaßcompiler erfordert allerdings eine Reihe zusätzlicher Transfers, was der Forderung nach hoher Compiliergeschwindigkeit zuwiderläuft.

Deshalb wurden so wenig wie möglich Pässe gewählt und darüberhinaus wurden logisch getrennte Aufgaben in ein und demselben Paß untergebracht (hier z.B. im Paß 5 die Anweisungsanalyse und die Erzeugung der umgek. Poln. Nation).

Ein weiterer Schritt in Richtung hohe Kompiliergeschwindigkeit war darauf gerichtet, eine möglichst geringe Belastung der Peripherspeichernahtstelle zu erreichen, d.h. die erforderlichen Plattentransfers auf ein notwendiges Minimum zu begrenzen. Dies wurde u.a. mit folgenden Maßnahmen erreicht:

- Absetzen von kurzen Zwischen-Codes in den einzelnen Pässen, durch Einträge variabler Länge (also eine kompakte Zwischensprache)
- Möglichst viel Informationen in sequentiell zu lesendem Zwischencode bearbeiten - also bearbeiten und vergessen! und nicht über Tabellen
- Einrichten eines eigenen Passes für den Listenaufbau, um einen großen Hauptspeicher-Listenbereich zu erhalten
- Den HSP-Listenbereich flexibel gestalten, um eine möglichst gute Ausnutzung des vorhandenen Laufbereiches zu erreichen
- Die Syntaxanalyse weitgehend sackgassenfrei ausführen, womit erreicht wird, daß der Input-Zeiger selten auf einen schon gelesenen Eintrag zurückzusetzen ist.

Schließlich wurden, um der Forderung "hohe Kompiliergeschwindigkeit und Einhalten eines Laufbereichs von max. 20 KW" möglichst nahe zu kommen, die Sprachen MECO und Assembler zur Implementierung benutzt.

MECO ist eine für Compilerbau, genauer für Syntaxanalyse, im Hause Siemens bereits mehrfach eingesetzte und bewährte Sprache. Als Ergebnis eines Kompiliervorganges geht es dem Anwender u.a. auch darum, exakte Hinweise auf den Ort eines Fehlers, verbunden mit einem aussagekräftigen Fehlertext für syntaktische Fehler, zu erhalten.

Bild 5 zeigt die Realisierung: Es werden, wo immer es geht, Fehlermeldungen im Quellspracheprotokoll am Ort des Fehlers eingefügt. Die erwähnten Fehlermeldungen erscheinen automatisch im Kompilierprotokoll solange Syntaxfehler vorhanden sind.

Für den Laufzeittest oder logischen Test kann man zusätzlich im Testmode übersetzen, d.h. man führt in diesem Fall eine zusätzliche Bedienung aus und erhält als Ergebnis ein PEARL-Adreßbuch, und im Grundspracheprotokoll wird zusätzlich Testhilfeinformation eingefügt (Bild 6). Letztere beinhaltet z. B. Zeilennummern, Informationen (Aufrufe) für das Testsystem, während das Adreßbuch die Variablennamen mit Adresse (blockspezifisch) enthält.

Mit der so erhaltenen Zusatzinformation und einem PEARL-Testsystem, das beim Binden der Grundsprache hinzugefügt wird, können im Dialog z.B. folgende Testfunktionen ausgeführt werden:

- Protokollierung durchlaufener Quellsprachezeilen, also ein Trace, und das Anhalten am Beginn vorwählbarer Zeilen
- Anhalten, wenn eine Variable einen voreingestellten Wert erreicht hat und
- Manipulation von Variablen mit Hilfe des im Prozeßrechnersystem 300-16 Bit enthaltenen Testsystems TEPOS (TEPOS im Bild nicht dargestellt).

Diesen Compiler, den ich Ihnen mit seinen charakteristischen Merkmalen und in seiner Systemprogrammumgebung kurz vorgestellt habe, wurde, wie bereits erwähnt, im Januar 78 freigegeben, er ist also seit nunmehr 2 1/2 Jahren im Einsatz.

In dieser Zeit haben wir 3 weitere Freigaben dieses Compilers herausgebracht (Bild 7), so daß nunmehr die Variante D vorliegt.

Es kann mehrere Gründe für weitere Freigaben eines SW-Produktes geben. Fehlerbeseitigungen, Funktionserweiterungen und/oder Verbesserungen sind wohl die häufigsten.

Da jedes Systemsoftwareprodukt einer gründlichen Abnahmekontrolle vor Freigabe unterworfen wird, wäre die Zahl der bisher aufgetretenen Fehler kein Grund für 3 weitere Freigaben gewesen, man

hätte das durch Nachführen beheben können. Funktionserweiterungen in Form von Spracherweiterungen haben wir nicht vorgenommen, da für die Akzeptanzererprobung der Basis-Subset uns ausreichend erschien. So haben wir den Schwerpunkt auf Verbesserungen, d.h. Optimierung des Compilers, gelegt.

Variante B - Umstellung auf R-Familie

- Feldzugriff: Ausnutzung der HW für wortbündige Feldelemente (z.B. FIXED, FLOAT) durch spezielle Codeerzeugung und Optimierung der Adreßberechnung für ein Feldelement.

Variante C - Laden des Laufzeitsystems in Common DATA-Bereich (ablaufinvariantes Laufzeitsystem)

- Binäre E/A: Ausnutzung des Wortrasters bei der Aufbereitung des E/A-Puffers und die Zusammenfassung von Verbunden (Extremfall ist Bit-Feld zu mehreren Worten à 16 Bit

Variante D - Adreßraumerweiterung: z.B. virtuelle Adressierung (PEARL C, d.h. globale Anwenderdaten in eigenem Laufbereich)

- Überarbeitung des gesamten Laufzeitsystems, z.B. unter Verwendung der mit der 300 R-Familie zur Verfügung stehenden HW-Befehle wie doppelt lange-, Gleitpunkt-, Byte-Befehle.

Bild 8 zeigt an Hand von zwei unterschiedlichen Beispielen die mit diesen Optimierungen erzielten, gemessenen Laufzeitverbesserungen.

Beispiel 1 Es handelt sich um ein rechenintensives Programm aus der Lastflußrechnung (eine gedrängt gespeicherte - da schwach besetzte - Matrix aus komplexen Zahlen wird invertiert).

Variante A lieferte gegenüber der ausprogrammierten Assemblerlösung ein um ca. 2,3-fach langsameres Ergebnis. Die Optimierung des Feldzugriffes reduzierte

diese Laufzeit auf den Faktor 1,4, und die des Laufzeitsystems (hier besonders die Routine zur Adreßberechnung eines Feldelements) reduzierte nochmals und ergab nun nur noch eine Verlangsamung um den Faktor 1,18 gegenüber der Assemblerlösung.

Beispiel 2 Binäres Transferieren eines 256 W langen Feldes:
Dieses Beispiel zeigt, daß nach der Optimierung der binären E/A die Transferzeit praktisch nur durch die mittlere HW-Zeit bestimmt wird;mittlere HW-Zeit:
= mittlere Zugriffszeit, Org-Zeit, eigentlicher Transfer.

Diese Ergebnisse sind jedoch nicht zu verallgemeinern. Ich möchte ausdrücklich betonen, daß diese äußerst positiven Ergebnisse nicht für alle Anwendungen zutreffen. So wirkt z.B. die Optimierung im letzten Beispiel dann nicht, wenn man skalare Größen transferiert.

Dennoch glauben wir, durch Einarbeiten eigener und Anwendererfahrungen inzwischen einen äußerst stabilen Compiler zu haben, der den gesteckten, zu Anfang meines Referates aufgezeigten Zielen weitgehend entspricht. (Bild 9).

Die Energieversorgung Ostbayerns, OBAG, war nun ihrerseits bereit, ein Pilotprojekt unter Verwendung von Basis-PEARL zu starten und die Tragfähigkeit der Sprache, aber auch die Leistungsfähigkeit unseres PEARL-Kompiliersystems zu erproben. Das PEARL-Kompiliersystem wurde hier von Anfang an eingesetzt und einige wesentliche Verbesserungen - z. B. die von mir gezeigte Transferoptimierung - wurden aufgrund der hier gemachten Erfahrungen durchgeführt und es hat sich bestätigt, daß sowohl die Sprache als auch die Compiler im Einsatz reifen müssen.

Ich möchte den späteren Ausführungen zwar nicht vorgreifen, doch darf ich schon soviel sagen: die gemachten Erfahrungen mit Basis-PEARL sind insgesamt sehr positiv und berechtigen dazu, den eingeschlagenen Weg fortzusetzen.

1969	Beginn der Sprachentwicklung
7/76	Fertigstellung eines Pilot-Compilers bei SIEMENS
9/77	Beschreibung von Basis-PEARL
1/78	Freigabe des ersten Basis-PEARL-Compilers durch SIEMENS, für Rechner SIEMENS 330

Bild 1 Entwicklung des PEARL-Compilers

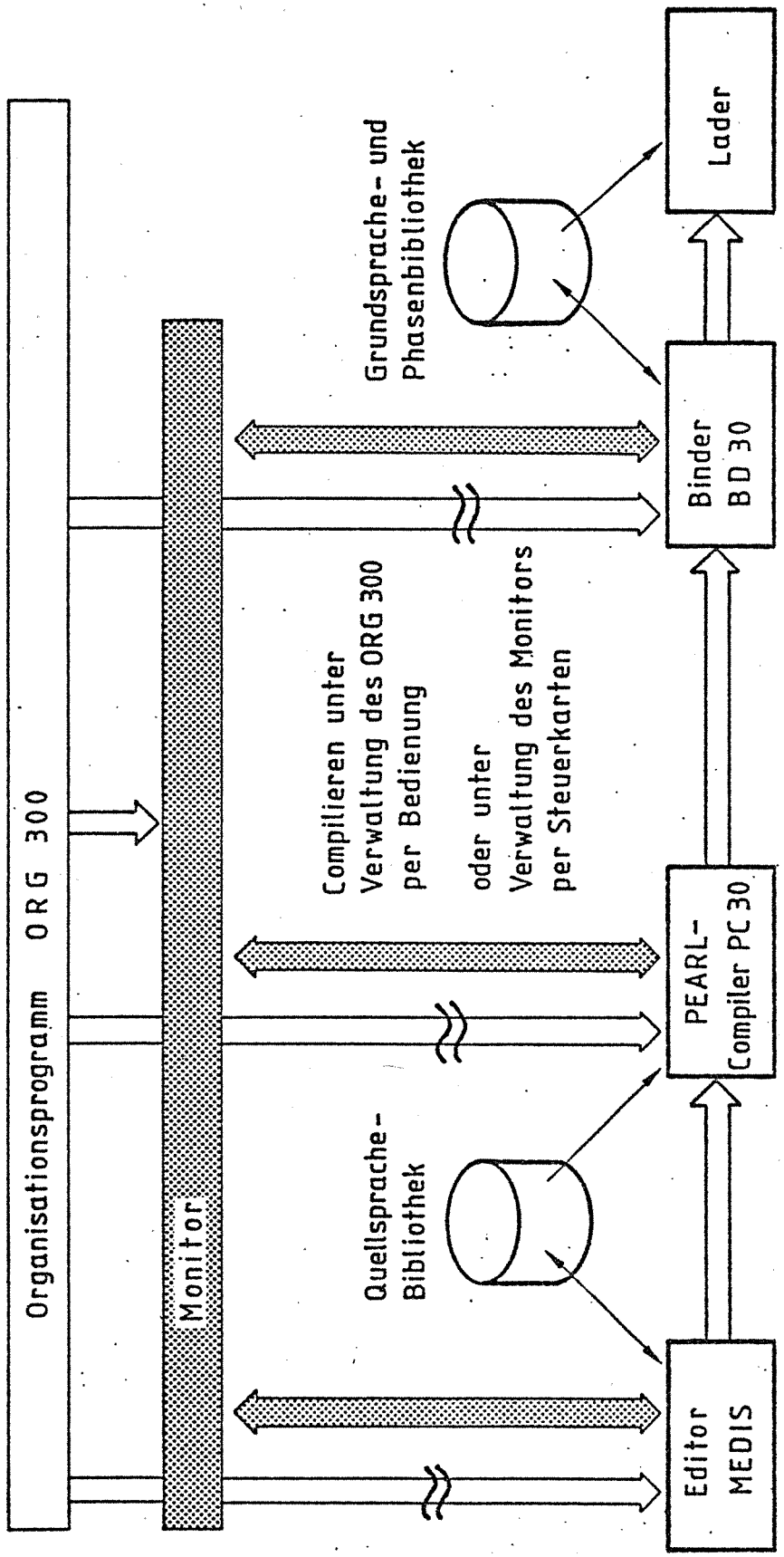
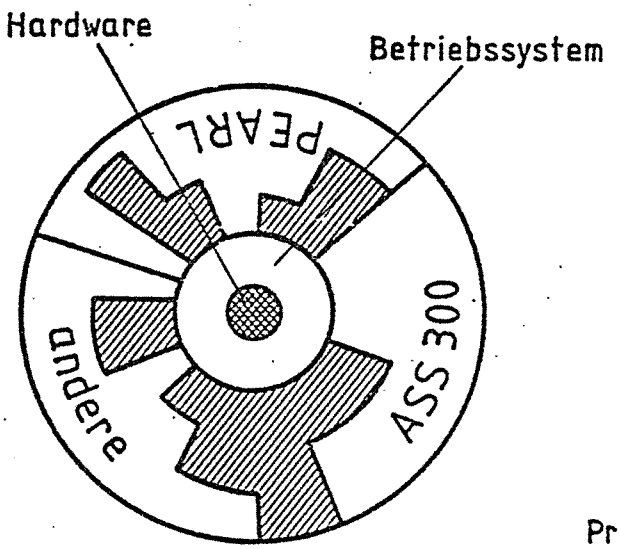
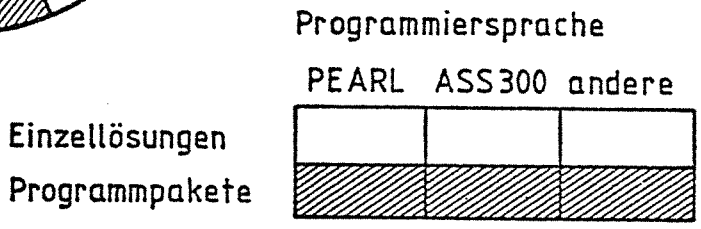


Bild 2 PEARL-Compiler im Spektrum der Dienstprogramme





PEARL-Programme
und
<u>andere Anwenderprogramme</u>
müssen

- sich gegenseitig starten
- sich gegenseitig koordinieren
- Daten austauschen
- gemeinsame Geräte benutzen
- sich bezüglich z.B. Wiederanlauf udgl. ähnlich verhalten

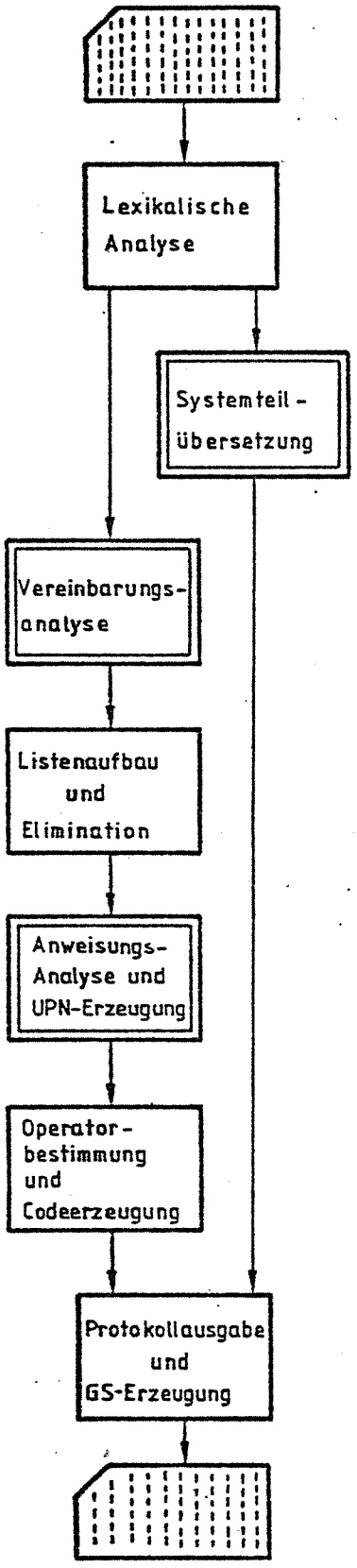
daher ⇒ Kein eigenes Betriebssystem für PEARL

PEARL-Programme benutzen ORG 300

PEARL-Programme
und
<u>Programmpakete</u>
müssen darüber hinaus

- über technologisch orientierte Datennahtstellen verkehren

Bild 3 Zusammenwirken PEARL-Programm-Umgebung



- Mehrpaßcompiler

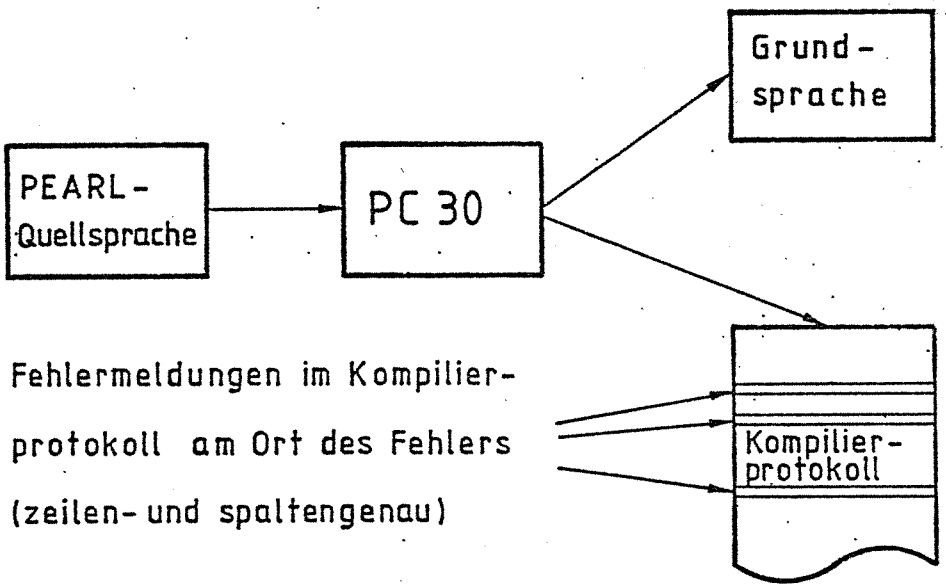
Trotzdem gute Kompiliergeschwindigkeit durch

- Wenig Pässe
- Geringe Belastung der Peripherspeichernahtstelle
- Geeignete Implementierungssprachen

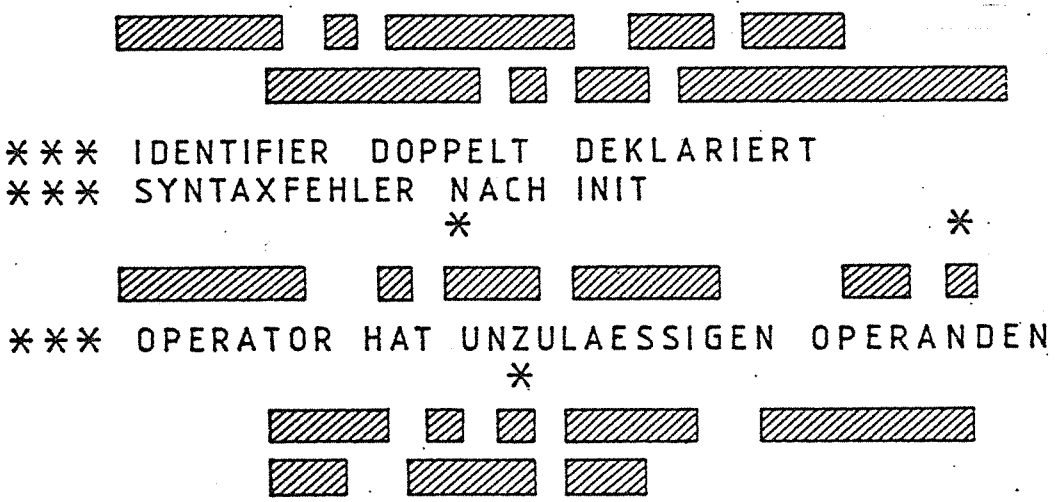
MECO für Syntaxanalyse

Assembler

Bild 4 Aufbau des PEARL-Compilers



Beispiel für Fehlermeldungen (schematisch)

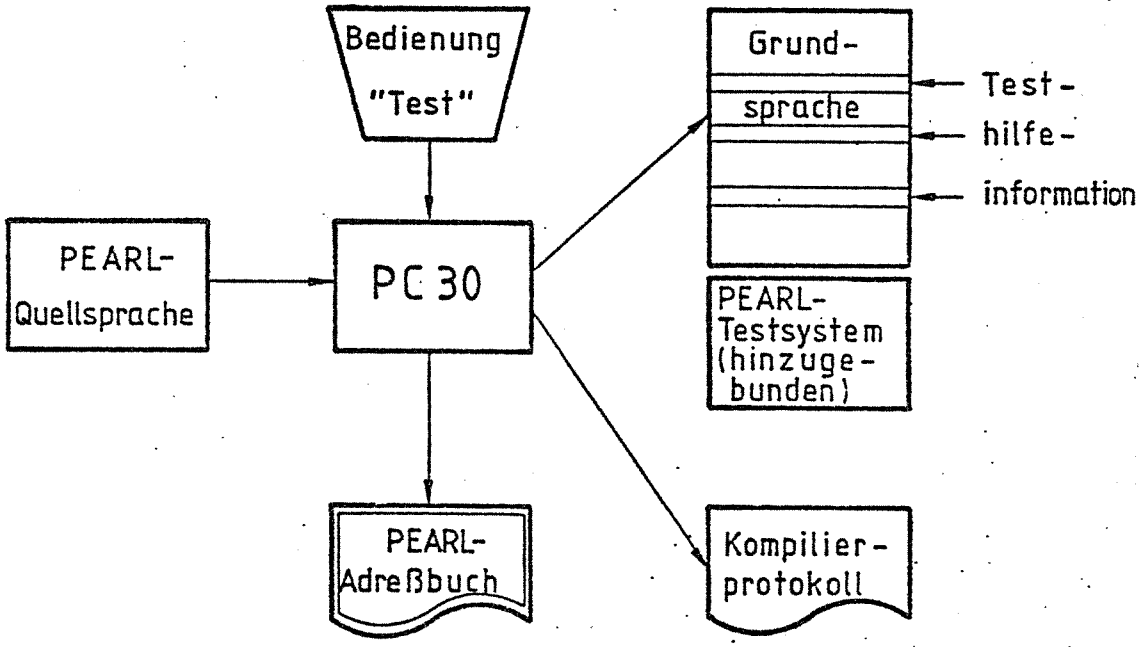


PEARL-Quelltext

* * * Fehlermeldezeilen (* in Folgezeile bezeichnet Spalte)

Bild 5 Fehlermeldungen des PEARL-Compilers

Übersetzen im Testmode



Testen im Dialog

Stufe 1: Mit Kompilierprotokoll und PEARL-Testsystem

- Trace (Protokollierung der Nummern durchlaufener Zeilen)
- Haltepunkte setzen (an Zeilenanfängen)

Stufe 2: Mit PEARL-Adreßbuch und PEARL-Testsystem

- Variablenkontrollstop (wertabhängig)

Stufe 3: Mit PEARL-Adreßbuch und Assembler-Testsystem TEPOS (nicht im Bild)

- Werte von Variablen überprüfen/ändern

Bild 6 Hilfsmittel für den Laufzeittest

Compiler seit $2\frac{1}{2}$ Jahren im Einsatz:

Variante	Besonderheit	Freigabe	Optimierung
A	für 330	1/78	
B	für R-Serie	1/79	Feldzugriff
C	Laden des Lauf-zeitsystems in CD	11/79	Binäre E/A
D	Adreßraumer-weiterung	4/80	Laufzeitsystem als Ganzes

Bild 7 Varianten des PEARL-Compilers

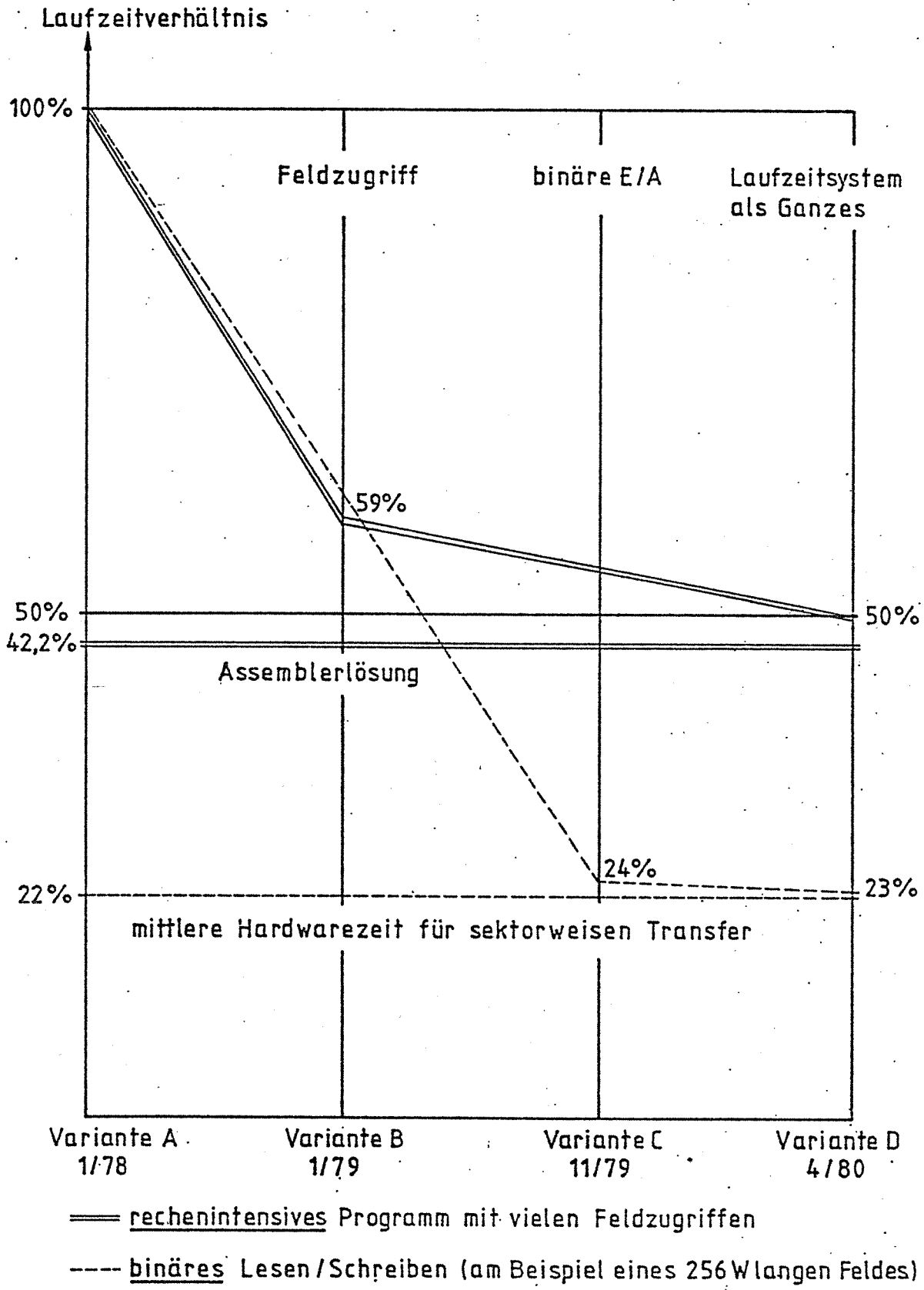


Bild 8 Laufzeitoptimierung

- Platzbedarf auf Platte

Ausgelieferter PC 30	190 kW
Ausgelieferte Laufzeitroutinen	65 kW
Vom Ladebinder erstellte Version (einfache Listenlänge: Page = 1)	195 kW

- Platzbedarf im HSP

Laufbereich für Compiler	≥ 17 kW
Generierbares ablaufinvariantes Laufzeitsystem	≤ 17 kW

- Lieferform

  Auf Platte, einzeln oder innerhalb Systempaket

- Übersetzungsgeschwindigkeit

  ~ 350 - 700 Zln/min

  (~ 4 kW Code/min)

Bild 9 Technische Daten des PEARL-Compilers