KERNFORSCHUNGSZENTRUM

KARLSRUHE

März 1975 KFK 2121

Institut für Angewandte Kernphysik

Das Metacompilersystem META-II/X
Implementierung eines CAMAC-IML-Precompilers

W. Kneis

GESELLSCHAFT
FÜR
KERNFORSCHUNG M.B.H.

KARLSRUHE

GESELLSCHAFT FÜR KERNFORSCHUNG M. B. H.

KARLSRUHE

KERNFORSCHUNGSZENTRUM KARLSRUHE

Februar 1975 KFK 2121

Institut fuer Angewandte Kernphysik

DAS METACOMPILERSYSTEM META-II/X

-

IMPLEMENTIERUNG EINES CAMAC-IML-PRECOMPILERS

von

Wilfried Kneis

Gesellschaft fuer Kernforschung m.b.H., Karlsruhe

ZUSAMMENFASSUNG

Das Ziel dieser Arbeit ist, anhand der Eigenschaften des META-II/X-Systems und einer konkreten Compilerimplementierung fuer IML zu zeigen, dass es mit einem einfachen und universell einsetzbaren Symbolprozessor sehr leicht moeglich ist, Precompiler fuer problemorientierte Sprachen zu erstellen. Hierbei steht die Tatsache im Vordergrund, dass keine handcodierten Hilfsroutinen fuer die spezielle Implementierung hinzugefuegt werden mussten. Die Uebersetzung von IML wird einzig und allein durch die in der Meta-Sprache abgefasste Compilerbeschreibung definiert.

Insgesamt erweist sich META-II/X als ein relativ leicht zu handhabendes System fuer die Uebersetzungsautomatisierung expliziter Sprachen. Entscheidend hierfuer ist die Wahl einer Assemblersprache als Objectsprache, wodurch nicht vollstaendig aufgeloeste Referenzen in die Assemblerebene uebertragen werden koennen. Die Implementierung beinhaltet die Moeglichkeit einer problemlosen Uebertragung des gesamten Systems inclusive der internen Compilerdarstellungen.

ABSTRACT

The Metacompiler System META-II/X - Implementation of a CAMAC IML Precompiler

It is the objective of this work to demonstrate by the properties of the META-II/X system and concrete compiler implementation for IML that a simple and universally applicable symbol processor allows to develop in a very easy manner precompilers for problem oriented languages. The main feature consists in the fact that no auxiliary routines coded manually had to be added for special implementation. The translation of IML is exclusively defined by the compiler description written in the META language.

As a whole, META-II/X proves to be a system which is relatively convenient to handle in automating the translation of explicit languages. The decisive point is the choise of an assembler language as target language allowing to transfer to the assembler level references not completely resolved. Implementation includes the possibility of an uncomplicated transfer of the whole system inclusive of the internal compiler representations.

INHALT

1 EINLEITUNG

Seit Erscheinen der grundlegenden Arbeiten von Chomsky (1), die von der grammatikalischen Erfassung einer Sprache ausgehen, hat sich in der Datenverarbeitung in immer staerkerem Masse der Trend zu einer Automatisierung des Uebersetzer-Schreibens durchgesetzt. Schon in der ersten Haelfte der sechziger Jahre existierten in den USA eine Vielzahl von Konzeptionen und Programmsystemen, sogenannte Translator-Writing-Systems (TWS), die in der Arbeit von Feldmann und Gries (2) zusammengefasst sind.

Die dem vorliegenden META-II/X-System (A-7) zugrundeliegende Konzeption, das META-II-Prinzip von D. V. Schorre (3), geht ebenfalls bis zum Jahre 1964 zurueck. Die allgemeine Behandlung der Compilererstellung als einer Symbol-Manipulations-Aufgabe gibt dem META-II/X-System teils mehr, teils weniger Kapazitaeten als sie die eigentlichen TWS's besitzen, die von einer gegebenen Grammatik ausgehen (2). Kapitel 2 dieser Arbeit beschaeftigt sich damit, gerade diese Kapazitaeten darzustellen. Unter dieser Einschraenkung (2) sind Systeme wie META-II/X als syntax-orientierte Symbolprozessoren in die Kategorie der TWS einzureihen.

Die vorliegende Arbeit ist zum einen Teil eine umfassende Beschreibung der Moeglichkeiten und Verfahren der Compilererstellung mit dem Metacompiler-System, META-II/X, und zum anderen Teil eine detaillierte Diskussion einer konkreten Implementierung eines Precompilers fuer die 'CAMAC intermediate level language', IML. Ausgangspunkt hierfuer war das Projekt der rechnergefuehrten Zyklotronkontrolle mit einem CDC 3100 Rechner ueber ein CAMAC-Multi-Branch-System (4) am Isochronzyklotron der GFK, Karlsruhe. Es bestand der Wunsch, die Anwendungsprogramme nicht in Maschinen-, sondern in einer CAMAC-spezifischen Sprache, d.h. einfach und uebersichtlich zu programmieren. Dazu musste die erforderliche Systemsoftware, ein Precompiler fuer eine CAMAC Sprache, selbst entwickelt werden (5). Was die CAMAC Sprache und ihren Sprachlevel anbetraf, so wurde das Konzept der ESONE-CAMAC-Software-Working-Gruppe (SWG), die 'CAMAC Intermediate Language' (IML), aufgegriffen. Die Gruende fuer diese Entscheidung waren in erster Linie, dass IML dem Wunsch nach schneller und leichter Implementierung entspricht, und zweitens, dass IML ein angehender internationaler Standard ist.

Auf der Basis der einfachen IML Syntaxstruktur (6) empfiehlt sich die Implementierung eines IML-Precompilers durch ein syntaxorientiertes 'Translator-Writing-System' (TWS). Wie im weiteren Verlauf der Arbeit gezeigt wird, koennen mit dem relativ einfachen System eines syntaxorientierten Symbolprozessors, META-II/X, saemtliche Funktionen bereitgestellt werden, um IML in eine Assemblersprache, CDC-COMPASS, (7) zu uebersetzen. Es wird ferner gezeigt, dass dies zum einen auf die einfache Syntaxstruktur von IML und zum anderen auf die Wahl einer Assemblersprache als Objectcode zurueckzufuehren ist.

2 META-II/X-SYSYEM

In der Gruppe der syntax-orientierten Symbolprozessoren zeichnet sich das von D. V. Schorre konzipierte META-II-System (3) durch leichte Anwendung, ausgezeichnetes Bootstrappingverfahren,

Erweiterungseigenschaften und eine relativ grosse Klasse beschreibbarer Sprachen aus. Im Gegensatz zu den meisten TWS geht die META-II-Konzeption davon aus, dass Parsing und Codegenerierung eng gekoppelt sind. Sie resultiert explizit darin, dass beide Prozesse durch eine Menge BNF-aehnlicher Regeln definiert werden. Die Regeln, die von einer durch sie dargestellten Satzstruktur ausgehen, stellen im Prinzip rekursive Recognizer mit eingebettetem semantischem Output dar. Diese strukturelle Kopplung von Syntax und Semantik ergibt eine Verknuepfung des Parsings mit der Codegenerierung. Weiterhin resultiert hieraus, dass META-II/X im allgemeinen als One-pass-Uebersetzer arbeitet, obwohl durch geeignete Formulierung auch Multi-pass-Parsing moeglich ist.

META-II/X bietet wesentliche Erweiterungen und weitere Eigenschaften gegenueber dem urspruenglichen META-II-System (3). Dies sind im einzelnen die Moeglichkeiten, Backups zu formulieren, und die Eigenschaft, symbolischen Output fuer Assemblersprachen zu produzieren (Abschnitte 2.2.2 und 2.3.2). Es unterscheidet sich gegenueber anderen Weiterentwicklungen von META-II (8,9,10,11) dadurch, dass versucht wurde, obige Erweiterungen einzugliedern, ohne die Klarheit und Uebersichtlichkeit der urspruenglichen Meta-Sprache des META-II-Systems aufzugeben.

2.1 Prinzip

Die Erstellung eines Compilers mit META-II/X besteht in der Aufgabe, die Definitionsregeln einer Sprache in der geeigneten Notation zu formulieren und mit Hilfe des Systems in ein ausfuehrbares Programm zu uebersetzen. Die fuer das META-II/X-System geeignete Notation ist die sogenannte Metasprache, ein BNF-aehnliches System von hierarchisch angeordneten Produktionsregeln, die Syntax und Semantik der zu definierenden Sprache beschreiben (A-6-1).

Um einen Compiler zu erzeugen, wird die im Input angebotene Metabeschreibung in eine interne Darstellungsform uebergefuehrt. Die interne Darstellungsform repraesentiert einen Compiler in der Form eines ausfuehrbaren Programmes. Dieses Programm wird von einem Systemunterprogramm, dem Meta-Simulator, ausgefuehrt. Die interne Darstellungsform wird daher als 'Meta-(Simulator-)Maschinencode' (A-2) bezeichnet. Der detaillierte Ablauf der Uebersetzung von Metabeschreibung in Meta-Maschinencode geschieht ueber die Zwischenstufe einer symbolischen Assemblersprache, der Meta-Assemblersprache (A-2).

Damit ergibt sich fuer eine Uebersetzung die folgende Beziehung zwischen den einzelnen Systemteilen:

(B1) CMS(SIM(CMC)) -> CMA(ASS) -> CMC

oder abgekuerzt:

(B1.1) CMS(SIM(CMC)+ASS) -> CMC

mit

MS	=	Meta-Sprache
MB	=	Meta-Beschreibung in MS
MA	=	Meta-Assemblersprache
MC	=	Meta-Maschinencode
SIM	=	Meta-Simulator(-Programm)
ASS	=	Meta-Assembler(-Programm)
SIM(MC)	=	Meta-Maschine (allgemein)
CMS	=	Compiler in MS
CMA	=	Compiler in MA
CMC	=	Compiler in MC

Obiger Ausdruck (B1) ist so zu interpretieren, dass, mit Hilfe der Meta-Maschine SIM(CMC) die Metabeschreibung CMS zunaechst in CMA uebersetzt wird. CMA wird anschliessend durch den Meta-Assembler ASS in die interne Darstellungsform CMC ueberfuehrt (Abb. 1).

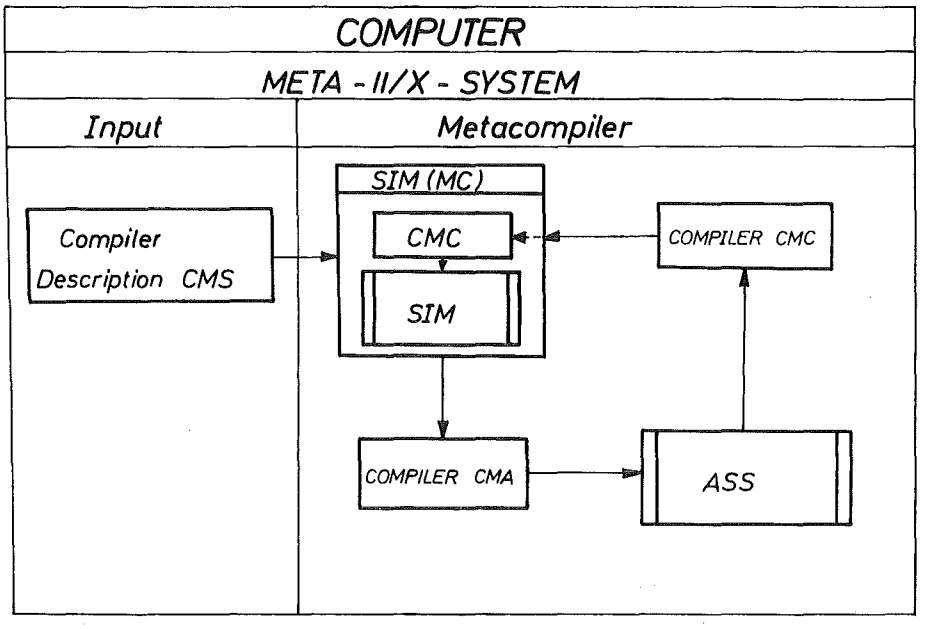


Abb. 1 Aufbau und Arbeitsweise des META-II/X-Systems

2.1.1 Selbstbeschreibung

Die Grundidee des META-II-Konzeptes basiert auf der Moeglichkeit, in MS einen Compiler zu formulieren, der die eigene Meta-Beschreibung uebersetzen kann, d.h. Selbstbeschreibungseigenschaften besitzt. Sei CMC0' eine geeignete Version, so kann dieser Vorgang folgendermassen dargestellt werden:

(B2) CMS(SIM(CMC0')+ASS) -> CMC

Durch Auswahl geeigneter Produktionsregeln kann eine Metabeschreibung CMS0 derart formuliert werden, dass damit ein Compiler CMC0 erzeugt werden kann, der, bis auf redundante Elemente, identisch mit der per Hand codierten Version CMC0' ist.
In Kurzform:

(B3) CMS0(SIM(CMC0')+ASS) -> CMC0

Ein weiterer Uebersetzungslauf von CMS0 mit SIM(CMC0),

(B4) CMS0(SIM(CMC0)+ASS) -> CMC0

kann den aus (B3) erhaltenen Compiler CMC0 reproduzieren.

Ein Beispiel der Formulierung geeigneter Regeln zur Selbstbeschreibung von CMC0 sei in der folgenden, nur von den Grundsymbolen,

(B5) NONTERMINAL, TERMINAL, LETTER, DIGIT

und den unmittelbar daraus abgeleiteten Primitivsymbolen bzw. speziellen Grundsymbolen,

(B6)
```
'     =  NONTERMINAL
.STR  =  'TERMINAL ...'
.ID   =  LETTER(LETTER/DIGIT) ...
```

ausgehende Metabeschreibung (B7) gegeben.

(B7)
```
PROG ::= $ STMT ;
STMT ::= .ID '::=' $ EXPR ';' ;
EXPR ::= ELMT $ ('/' ELMT ) ;
ELMT ::= .ID/.STR/'.ID'/'.STR'/
         '(' EXPR ')' / '$' EXPR ;
```

Diese Beschreibung (B7) besitzt zwar keinen Semantikoutput und kann daher nicht unmittelbar fuer den Selbstbeschreibungszyklus von META-II/X eingesetzt werden (vgl hierzu A-6-1); sie genuegt aber der fuer eine Selbstbeschreibung notwendigen syntaktischen Voraussetzung. Diese, in (B7) enthaltene Voraussetzung kann explizit so formuliert werden, dass alle, auf der rechten Seite einer Produktionsregel erscheinenden Terme entweder

(B7.1)
- Grundsymbole (B5)
- aus Grundsymbolen abgeleitete Primitivsymbole (B6)
- oder Aufrufe an andere, in der Metabeschreibung enthaltenen Produktionsregeln (B7)

sein muessen.

2.1.2 Erweiterung

Der Compiler CMC0 kann dadurch erweitert werden, dass der Beschreibung CMS0 eine Menge CMS' weiterer Produktionsregeln oder Alternativen von Produktionsregeln (A-6-2) hinzugefuegt werden:

(B8) CMS1 = CMS0 + CMS'

Nach der Uebersetzung von CMS1

(B9) CMS1(SIM(CMC0)+ASS) -> CMC1

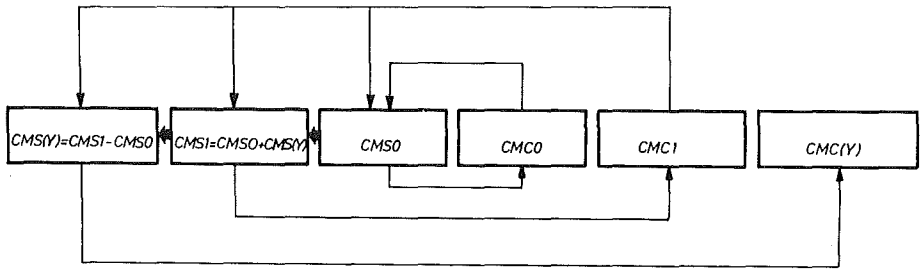
erhaelt man einen Compiler, CMC1, der ebenfalls seine eigene Beschreibung uebersetzen kann; d.h. auch der Uebersetzungsvorgang:

(C1) CMS1(SIM(CMC1)+ASS) -> CMC1

liefert den Compiler CMC1 (Abb. 2). Ueberdies kann mit CMC1 auch die urspruengliche Metabeschreibung, CMS0, uebersetzt werden:

(C2) CMS0(SIM(CMC1)+ASS) -> CMC0

Die wichtigste, (C1) und (C2) zugrundeliegende Eigenschaft ist, dass nur solche Erweiterungen vorgenommen werden, die entweder redundant oder invariant gegenueber der Selbstbeschreibungseigenschaft sind (B7.1).



CMS0 - Metabeschreibung von CMC0
CMC0 - Compiler für CMS0
CMS1 - Metabeschreibung von CMC1
CMC1 - Compiler für CMS1
CMS(Y) - Beschreibung der Sprache Y
CMC(Y) - Compiler für Sprache Y

Abb. 2 Erweiterungszyklus in META-II/X

Das folgende Beispiel mit dem direkt aus Grundsymbolen abgeleiteten Primitivsymbol

(C3) .NUM = DIGIT ...

sei als Erweiterung von (B7) unter Erhaltung der syntaktischen Selbstbeschreibungseigenschaft angefuehrt.

```
(C4)      PROG ::= $ STMT ;
          STMT ::= .ID '::='  $ EXPR ;
          EXPR ::= ELMT ( $('/'ELMT) / $<'/'ELMT> ) ;
          ELMT ::= .ID/.STR/.NUM/'.ID'/'.STR'/'.NUM'/
                   '('EXPR')'/'<'EXPR'>'/ '$' EXPR ;
```

(C4) erweitert die Beschreibung (B7) um die Moeglichkeit der Verwendung von ganzen Zahlen und eckigen Klammern, '<','>'. (Ein Beispiel mit Semantikoutput ist in A-6-1 gegeben.)
Wie im weiteren Verlauf dieser Arbeit gezeigt wird, eroeffnet die Erweiterungsfaehigkeit des META-II/X-Systems damit prinzipiell folgende Moeglichkeiten:

(C5)
- Bootstrapping
- Aenderung der Metabeschreibung
- Definition anderer Sprachen

2.2 Metacompiler

META-II/X ist physikalisch in drei Einheiten, den Kommandoprozessor, den Meta-Simulator SIM und den Meta-Assembler ASS, eingeteilt. Das gesamte System ist in FORTRAN IV geschrieben. Der Kernspeicherbedarf des FORTRAN IV Programms betraegt auf einer IBM/370 ungefaehr 24 k Bytes ohne Datenbereiche fuer Compiler, Stack-, System-, und Programm-Buffer. (Weitere Systemwerte sind A-5 zu entnehmen.) Die logische Einteilung (Abb. 1) geschieht zweckmaessigerweise in Kommandoprozessor (Input) und Metacompiler, der wiederum aus der Meta-Maschine SIM(MC) und ASS besteht. Wie aus Abb. 1 zu entnehmen ist, setzt sich SIM(MC) aus SIM und dem Compiler CMC zusammen.
Der Kommandoprozessor hat im wesentlichen die Aufgabe, anhand der Steuerkommandos (siehe A-1) den Arbeitsablauf bzw. die Betriebsart des Metacompilers zu ermitteln und zu steuern. Ausserdem unterstuetzt er den Input/Output des Metacompilers, wie z.B. Laden oder Ablagern der in MA oder MC vorliegenden bzw. produzierten Compilerversionen.

Die Betriebsarten des Metacompilers sind:

(C6)
1. Compilerlauf
2. Assemblerlauf
3. Compiler- + Assemblerlauf

Im Compilerlauf (1.) wird das im Input angebotene Programm in eine durch den Compiler CMC festgelegte Assemblersprache uebersetzt. Ist der Programminput die Metabeschreibung eines Compilers, so wird der Output von SIM zwangslaeufig in MA sein. Ein in MA vorliegender Compiler wird im Assemblerlauf von ASS in MC uebersetzt. Dies kann entweder in einem getrennten Lauf (2.) oder unmittelbar im Anschluss an den Compilerlauf (3.) geschehen. Durch Betriebsart (2.) ist die Transportabilitaet eines Compilers gesichert , da MA von der speziellen Byte-Struktur des ausfuehrenden Rechners unabhaengig ist (im Gegensatz zur internen Darstellung in MC).

Um eine im Input angebotene Sprachbeschreibung oder ein in einer

bestehenden Sprache geschriebenes Programm zu uebersetzen, wird vom Kommandoprozessor der entsprechende CMC-Compiler geladen und die Programmkontrolle an SIM uebergeben.

SIM besteht aus einer Reihe von Grundfunktionen (A-7, A-2) und einem zentralen Steuerteil in der Form einer FORTRAN-'GOTO'-Anweisung. Die Menge der Grundfunktionen besteht aus einer Reihe von Rekursivaufrufen, Verzweigungs-, Test- und Output-Instruktionen. Der Ablauf der Uebersetzung wird von dem CMC-Compiler mit Hilfe des zentralen SIM-Steuerteils bestimmt und kontrolliert. Jede Grundfunktion erhaelt vom SIM-Steuerteil die Kontrolle und gibt sie nach erfolgter Operation wieder an ihn zurueck.

SIM(MC) ist im wesentlichen dadurch charakterisiert, dass sie dem Uebersetzungsvorgang einen internen Wahr/Falsch-Schalter S zuordnet. Ein Programm ist damit genau dann richtig uebersetzt, wenn am Ende der Uebersetzung der Schalter S den Wert 'wahr' besitzt. Die Grundfunktionen von SIM(MC) koennen bezueglich S prinzipiell in zwei Gruppen - die der aktiven und die der passiven Funktionen - eingeteilt werden. Die Funktionen passiver Art veraendern den Wert von S nicht, sondern machen entweder ihre Aktion von dem Wert von S abhaengig (Verzweigungsinstruktionen) oder operieren unabhaengig von dem Wert von S (Outputinstruktionen und Rekursivaufrufe). Die aktiven Funktionen - alle Testfunktionen - veraendern den Wert von S in Abhaengigkeit eines im Input getesteten Ausdruckes. SIM(MC) besitzt drei verschiedene Arten von Testfunktionen:

1. Test fuer Identifier
2. Test fuer Strings
3. Test fuer spezielle Strings.

ASS uebersetzt den von SIM(MC) ausgegebenen MA in die interne Darstellung MC. Er arbeitet als Two-Pass-Assembler, der im ersten Pass anhand der verschiedenen Instruktionsformate die Programmlaenge und die Symboladressen festlegt und im zweiten Pass die Codegenerierung mit MC als Objectsprache vornimmt.

2.2.1 Notation

Die in MS abgefasste Meta-Beschreibung eines Compilers oder einer Sprache besteht aus einer Reihe von Syntaxgleichungen bzw. Produktionsregeln. Sie definieren ein hierarchisch strukturiertes System, das Top-Down-Parsing ermoeglicht. Die linke Seite einer Syntaxgleichung besteht aus dem Namen (metalinguistische Variable) der Konstruktion, die auf der rechten Seite definiert wird.

Ausgehend von (B5) und (B6) gilt folgende Konvention (3):

1. Terminale Symbole werden in Hochkommas eingebettet.
2. Systemsymbole beginnen mit '.'.
3. Metalinguistische Variable werden als Identifier geschrieben.
4. Aufeinanderfolgende Ausdruecke bedeuten Konkatenation.
5. Alternativen werden durch '/' angezeigt.
6. Klammer sind vorgesehen, um die Prioritaet zwischen

Alternation und Konkatenation zu regeln; (im Normalfall besitzt die Konkatenation eine hoehere Prioritaet gegenueber Alternation.)

Die in A-6-1 aufgefuehrte Meta-Beschreibung CMS0 ist selbstbeschreibend, d.h. sie definiert einen Compiler CMC0, der seine eigene Beschreibung (CMS0) uebersetzen kann (B4). Es ist die einfachste Version eines solchen Compilers bzw. der dazugehoerigen Metabeschreibung. Die folgenden Ausfuehrungen sollen die Interpretation der Meta-Beschreibung CMS0 erlaeutern (A-6-1, A-3, A-4).

.SYNTAX PRO definiert das Hauptelement von CMS0, die Syntaxgleichung mit dem Namen PRO.

PRO = ... bedeutet, dass ein Programm mit .SYNTAX beginnt und von einem Identifier gefolgt wird. Der Identifier wird als Label ausgegeben. Nachfolgende Elemente (angedeutet durch $ ST) koennen null oder mehrere Produktionsregeln in der Form einer Syntaxgleichung sein (Aufruf der Regel ST). Das Ende eines Programmes wird durch .END angedeutet. Das String 'END' wird als Operationscode ausgegeben.

ST = ... beschreibt den Aufbau einer Syntaxgleichung. Eine Syntaxgleichung beginnt mit einem Identifier, dem Namen der betreffenden Konstruktion. Der Identifier wird als Label ausgegeben. Das naechste Zeichen muss ein Gleichheitszeichen sein. Die rechte Seite einer Syntaxgleichung wird durch einen Ausdruck der Form EX1 beschrieben. Das Ende der Produktionsregel wird durch '.,' angedeutet. Wird ein '.,' im Input entdeckt, so wird der Operationscode R fuer die rekursive Rueckkehr in die aufrufende Konstruktion ausgegeben.

EX1 = ... verweist unmittelbar auf die Produktionsregel EX2. Ausdruecke der Form EX1 sind dadurch charakterisiert, dass sie die Moeglichkeit der Alternativbildung beschreiben; sie koennen Alternativen ('/') der Form EX2 enthalten (angedeutet durch den Prefixoperator $). Wird eine Alternative entdeckt, so wird der Befehl 'BT *1' ausgegeben, wobei '*1' den aktuellen Label darstellt. Sind keine Alternativen mehr vorhanden, so wird der aktuelle Label in das Labelfeld eingetragen.

EX2 = ... beschreibt die Konkatenation von Ausdruecken der Form EX3 oder OUT. EX2 verweist zuerst auf die Produktionsregel EX3 oder OUT. Falls EX3 zutrifft, wird die Instruktion 'BF *1' ausgegeben. $(...) laesst zu, dass noch weitere Terme der Form EX3 oder OUT auftreten duerfen. Im Fall von weiteren Termen der Form EX3 wird die Instruktion 'BE' ausgegeben. Ist dies nicht mehr zutreffend, so wird der aktuelle Label ins Labelfeld eingetragen.

EX3 = ... beschreibt Primitivsymbole (B6) der Form .ID und .STRING (Nach .ID wird der Rekursivaufruf 'CLL *' und nach .STRING der Operationscode 'TST *' fuer das Testen eines gegebenen Strings ausgegeben.), die Spezialstrings '.ID' und '.STRING' (Ausgabe der Operationscodes 'ID' fuer den Test auf Identifier und von 'SR', um zu testen, ob ein String von ' begrenzt wird), Ausdruecke der Form EX1 (wenn diese von runden Klammern begrenzt sind) oder schliesslich ein $-Zeichen. Wird ein $-Zeichen im Input erkannt, so erfolgt die Ausgabe des aktuellen Labels in das Labelfeld , und es

koennen weitere Ausdruecke der Form EX3 folgen. Ist dies nicht der Fall, so werden abschliessend die Instruktionen 'BT *1' und 'SET' ausgegeben. Diese Sequenz bedeutet, dass der Rekursivaufruf von EX3 genau dann beendet wird, wenn kein Ausdruck der Form EX3 mehr folgt.

OUT = ... beschreibt die semantischen Outputanweisungen. Die Syntax dieser Befehle ist gegeben durch: 1.) .OUT, gefolgt von mehreren, in Klammern eingeschlossenen Ausdruecken der Form OU1 oder 2.) durch .LABEL, verbunden mit der Ausgabe der Instruktion 'LB'. Der nachfolgende Ausdruck muss der Form OU1 entsprechen. Abgeschlossen wird diese Syntaxgleichung mit der Ausgabe der Instruktion 'OUT' (nicht zu verwechseln mit dem Namen der aktuellen Syntaxgleichung).

OU1 = ... beschreibt Argumente der semantischen Anweisungen .OUT und .LABEL. Als Argumente sind entweder eines der Spezialstrings *1, * oder ein allgemeines String (angedeutet durch .STRING) zugelassen. Im Fall von *1 wird ein aktueller Systemlabel generiert (durch Ausgabe des Befehls 'GN 1'), bei * wird der Input kopiert ('CI') und bei .STRING wird ein durch den Befehl 'CL *' vorgebenes String kopiert.

2.2.2 Eigenschaften

Der Parsingalgorithmus von SIM(MC) geht von dem rechtsrekursiven Aufbau der Syntaxgleichungen aus,

```
(C7)        EX1 ::= EX2 (EX1 / .EMPTY) ;
            EX2 ::= .TEST ;
```

d.h., eine Metabeschreibung CMS darf keine linksrekursive Regeln der Form

```
(C8)        EX1 ::= (EX1 / .EMPTY) EX2 ;
            EX2 ::= .TEST ;
```

enthalten, wobei .TEST eine beliebige Testfunktion markiert, und .EMPTY die Testfunktion fuer das Nullstring bedeutet (.EMPTY muss vorhanden sein, um die Rekursion abbrechen zu koennen.).

Eine abkuerzende Schreibweise fuer die in (C7) auftretende Konstruktion (EX1 / .EMPTY) ist durch die Einfuehrung des Prefixoperators $ gegeben; mit (C9) kann (C7) einfacher dargestellt werden.

```
(C9)        EX1 ::= EX2 $ EX1 ;
            EX2 ::= .TEST ;
```

Der Prefixoperator $ erlaubt, dass der ihm folgende Ausdruck null- bis n-mal (wobei n nicht fest vorgegeben ist) auftreten darf, d.h., auch 'optional' sein kann. So bewirken z.B. in A-6-1 die in der Produktionsregel EX3 dem EX3-Aufruf folgenden Outputanweisungen .OUT('BT' *1) und OUT('SET') einen solchen fehlerfreien Abbruch der durch die Syntaxgleichungen EX1, EX2 und EX3 aufgebauten Rekursion.

Aufgrund des rechtsrekursiven, backup-freien Parsingalgorithmus der

Meta-Maschine SIM(CMC0) (A-6-1) ist es notwendig, die Produktionsregeln so zu formulieren, dass spezielle Alternativen vor den sie enthaltenden allgemeinen Alternativen abgeprueft werden. In dem folgenden Beispiel (D1) mit der Regel EX, die die Inputstrings 'LABEL *1' und .ID '*' wobei .ID einen Identifier beschreibt – zulassen sollte, wird fuer die Stringfolge 'LABEL *1' die zweite, richtige Alternative nie erreicht, da das System in der ersten Alternative bei der Pruefung auf '*1' mit einem Fehlerstop endet.

```
(D1)        EX = .ID '*' .OUT('IDENTIFIER') /
                 'LABEL' '*1' .OUT('LABEL') ;
```

Die richtige Formulierung fuer (D1) muss folgendermassen lauten:

```
(D2)        EX = 'LABEL' '*1' .OUT('LABEL') /
                 .ID '*' .OUT('IDENTIFIER') ;
```

Die Beendigung des Parsings nach (D1) durch Fehlerstop folgt zwangslaeufig aus der Tatsache, dass SIM(CMC0) nicht die Moeglichkeit besitzt, Backups vorzunehmen. Der hierfuer verantwortliche Term .OUT('BE') aus A-6-1 befindet sich in der Produktionsregel EX2. Um Backups zuzulassen, muessen an dieser Stelle Anweisungen vorgesehen werden, die ein Zurueckspulen des bisher analysierten Inputs bzw. des bisher ausgegebenen Outputs ermoeglichen, falls eine Fehlerbedingung auftaucht. Wenn die Backup-Moeglichkeit fuer SIM(CMC0) nicht explizit programmiert wurde, bedeutet dies, dass die Syntaxgleichungen so aufgebaut sein muessen, dass die zu Beginn einer Produktionsalternative stehende, obligatorische Testfunktion entscheidet, ob die durch sie charakterisierte Alternative fuer das weitere Parsing des Inputstrings zutreffend ist. Ist letzteres der Fall, und wird in der folgenden Analyse ein weiterer Ausdruck im Inputstring nicht mehr erkannt, so endet das System mit einem Fehlerstop bzw. mit einer Error-Diagnostic-Routine.

Die Frage, Backup oder Nicht-Backup hat Schorre in (3) weitgehend diskutiert. Die Hauptargumente fuer Backup bzw. Nicht-Backup sind hiernach, dass einerseits die Klasse der Sprachen, die nur mit Backup-Moeglichkeiten beschrieben werden koennen, groesser ist als die Klasse der durch backup-freie Systeme beschreibbarer Sprachen. Andererseits kann jedoch festgestellt werden, dass in vielen Faellen durch entsprechende Formulierung Backups vermieden (D2) bzw. simuliert (D3) werden koennen, sodass es durchaus sinnvoll scheint, auch backup-freie Systeme einzusetzen, zumal diese gegenueber Backup-Systemen den Vorteil des schnelleren Parsings besitzen.

Soll z.B. in (D2) auch das Inputstring 'LABEL *' zugelassen, aber als Identifier klassifiziert werden, so kann ein hierzu notwendiges Backup durch folgende Formulierung simuliert werden.

```
(D3)        EX ::= 'LABEL' ('*1' .OUT('LABEL') / EX) /
                   (.ID / .EMPTY) '*' .OUT('IDENTIFIER') ;
```

Eine Grenze fuer diese Simulation setzt die daraus resultierende Unuebersichtlichkeit der Meta-Beschreibung bzw. die durch die Nullstring-Testfunktion .EMPTY involvierte Komplexitaet der Beschreibung. Ausserdem ergibt sich in vielen Faellen eine unter Umstaenden nicht gewuenschte Erweiterung der zu uebersetzenden Sprache. So laesst z.B. (D3) neben den gueltigen Stringfolgen 'LABEL *1', .ID '*'

(darunter auch 'LABEL *') weitere Stringfolgen der Form, 'LABEL' .ID '*', zu. Die folgende Formulierung (D4) hingegen verbietet Stringfolgen obiger Art.

```
(D4)        EX ::= '*' .OUT('IDENTIFIER') /
                   'LABEL' ( '*1' .OUT('LABEL') / EX) /
                   .ID '*' .OUT('IDENTIFIER') /
                   .EMPTY ;
```

2.2.3 Backup

Die in META-II/X implementierte Backup-Moeglichkeit ist dadurch charakterisiert, dass Backups nur dort durchgefuehrt werden, wo dies in der Compilerbeschreibung explizit aufgefuehrt ist. Dadurch wird erreicht, dass der Backupschalter 'UP' nur lokal wirkt. Das Prinzip der Implementierung wird in Abb. 3 anhand eines Beispiels (entnommen aus der Uebersetzung der Beschreibung in A-6-4) verdeutlicht.

```
        a)                b)                c)

      ID                ID                ID
      UP
      BF  A01           BF  A01           BF  A01
    TST x:x           TST x:x           TST x:x
      BE                BE                BE
      BF  A02           BF  A02
      OUT               OUT               OUT
A02                 A02
A01 BT  A03       A01 BT  A03       A01 BT  A03
      ...               ...               ...
```

Abb. 3 Illustration der Backup Loesung
a) Backup-Mode, Backup selektiert
b) Backup-Mode, Backup nicht selektiert
c) backup-freier Mode

Ein Vergleich der Versionen b) und c) zeigt, dass fuer backupfreie Probleme Version c) die geeignetere Loesung ist, da sie wesentlich kuerzer und daher effizienter ist. Eine detaillierte Beschreibung eines Beispiels mit Backup kann Anhang A-6-4 entnommen werden.

2.3 Zustandsarten

Ausgehend von den in Kapitel 2.2 aufgefuehrten Betriebsarten (C6) des Metacompilers ergeben sich hinsichtlich der Uebersetzung einer Sprache folgende Zustandsarten. Sie unterscheiden sich dadurch, dass der Metacompiler - je nach Zustandsart - als Interpreter, Compiler oder

Symbolprozessor in Erscheinung tritt. Da der Interpreter- und Symbolprozessorzustand von Systemen wie META-II (3), Meta-3 (8) bzw. Meta5 (9) benutzt wird, sollen im folgenden nur die wesentlichen Eigenschaften dieser Zustandsarten vermittelt werden. Der Compilerzustand wird hingegen ausfuehrlich diskutiert, da er die Grundlagen fuer den in Kapitel 3 dargestellten IML-Compiler liefert.

2.3.1 Interpreter

Der Interpreterzustand ist dadurch charakterisiert, dass der von SIM(MC) produzierte Objectcode in MA vorliegt, von ASS in MC uebersetzt und anschliessend von SIM interpretiert wird. Insgesamt ergibt sich damit die Uebersetzung einer Sprache L durch

(D5) L -> MA -> MC

Eine notwendige Voraussetzung hierzu ist die Erweiterung von ASS und die Emulation des entsprechenden Rechners durch SIM. Das bedeutet, dass zunaechst die benoetigten Elementarfunktionen semantisch definiert und in SIM incorporiert werden muessen. Anschliessend muss durch Erweiterung von ASS die Festlegung der entsprechenden Mnemonics erfolgen. Die Uebersetzung der Sprache L in MA erfordert daraufhin lediglich die Formulierung eines geeigneten Compilers CMS(L).

Eine fuer die Emulation der Simulatorfunktionen grundlegende Eigenschaft des META-II/X-Systems ist die Umwandlung von Infixtermen in Postfixterme. Dies wird z.B. durch die von Schorre (3) vorgeschlagene und in Abschnitt 2.3.2 dargestellte Formulierung (D6) erreicht. Wenn man von der hieraus resultierenden Postfixnotation ausgeht, ergibt sich zwangslaeufig, dass die Gesamtheit der in SIM zu emulierenden Funktionen einen Push-Down-Automaten darstellt. Weiterhin folgt aus (D6) und (D7), dass die fuer Register-Automaten notwendige Umwandlung von Postfixtermen in sequentiellen Code in diesem Fall ueberfluessig ist und nur zusaetzlichen Uebersetzungsaufwand mit sich bringt. Im Falle des Interpreterzustands sollte daher in jedem Fall die Realisierung eines Push-Down-Automaten angestrebt werden. Der Grund hierfuer ist darin zu sehen, dass die Unterschiede in Ausfuehrungszeit und Kernspeicherbedarf von Push-Down-Automaten gegenueber der Ersparnis an Uebersetzungsaufwand kaum ins Gewicht fallen.

Ein weiterer Nachteil des Interpreterzustands ist darin zu sehen, dass SIM zum einen Teil aus reinen Uebersetzungsfunktionen und zum anderen Teil aus reinen Ausfuehrungsfunktionen besteht. Dies fuehrt in der gegenwaertigen, overlay-freien Implementierung zu einem erheblichen 'Overhead' bezueglich des Kernspeicherbedarfs sowohl zur Uebersetzungszeit als auch zur Interpretationszeit.

2.3.2 Compiler

Der Compilerzustand ist dann gegeben, wenn die zu uebersetzende Sprache L nicht in MA uebersetzt wird, sondern der von der geeigneten Meta-Maschine SIM(CMC(L)) produzierte Objectcode die Assemblersprache

eines reellen Rechners ist. In diesem Fall muessen in SIM nur die Funktionen bereitgestellt werden, die zur Uebersetzung der Sprache L und nicht zur Ausfuehrung eines in L geschriebenen Programmes dienen (3.2). Damit besteht SIM nur aus der Menge der benoetigten Uebersetzungsfunktionen. Im Gegensatz zum Interpreterzustand ergibt sich hier kein 'Overhead' bezueglich des Kernspeicherbedarfes waehrend der Uebersetzungszeit.

Bei der Uebersetzung von arithmetischen Ausdruecken ist bei META-II/X zunaechst davon auszugehen, dass Infixterme in Postfixterme umgewandelt werden. Dies ist eine zentrale Eigenschaft der urspruenglichen META-II-Konzeption (3), der auch das META-II/X-System zugrundeliegt. Die Umwandlung von Infixtermen in Postfixterme geschieht daher auch mit relativ wenig Aufwand in der Compilerbeschreibung (D6). Wenn hingegen sequentieller Code benoetigt wird, kann dies prinzipiell auf zwei Arten realisiert werden. Zum einen durch eine erhebliche Aenderung der Meta-Beschreibung eines Compilers und damit auch des Simulators SIM, um eine Umwandlung von Infixtermen in sequentiellen Code direkt – in einem Schritt – auszufuehren. Diese Loesung wurde in den auf META-II (3) aufbauenden Systemen Meta-3 (8) und Meta5 (9) verfolgt. Das Resultat war mit einem erheblichen Aufwand in Meta-Maschine und der dazugehoerigen Meta-Beschreibung verbunden. Der andere, in der gegenwaertigen Implementierung eingeschlagene Weg besteht darin, eine Uebersetzung von Infixtermen in sequentiellen Code in zwei Schritten zu vollziehen. Zunaechst werden Infixterme in Postfixterme uebersetzt, in einen Stack gebracht und der Stackinhalt anschliessend in sequentiellen Code umgewandelt.

Der erste Schritt einer solchen Uebersetzung kann z.B. durch folgenden Algorithmus erreicht werden.

```
(D6)  ST  ::= ST1 '=' ('-' .PST('=00')/.EMPTY) EX1 .PST('.SBA')
              POP ;
      ST1 ::= .ID/ ... ;
      EX1 ::= EX2 $ ('+' EX2 .PST('.ADA')) ;
      EX2 ::= EX3 $ ('*' EX3 .PST('.MUA')) ;
      EX3 ::= .ID .PST(*)/.NUM .PST('=D' *)/'(' EX1 ')' ;
      POP ::= .EMPTY ;
```

Die hieraus resultierende Postfixnotation kann entweder von einem geeigneten Rechner (mit Push-Down-Hardware) direkt ausgefuehrt werden, oder in einem weiteren Step (wenn POP durch (D7) definiert wird) durch folgenden Algorithmus in sequentiellen Code umgewandelt werden.

```
(D7)  POP  ::= .FINDOP .POUT('LDA' 2) .POUT(0 1) $ STCK ;
      STCK ::= .OP .POUT(0 1) $ STCK /
               .POUT('STA' .HV) $ POP ;
```

Diese Loesung beinhaltet dieselben Uebersetzungskapazitaeten wie Meta-3 (8) und Meta5 (9) hinsichtlich der Umwandlung von Infixtermen in sequentiellen Code. Sie besitzt gegenueber diesen jedoch erhebliche Vorteile. Da die Umwandlung von Infixtermen nicht direkt durchgefuehrt wird, kann, nur durch formale Aenderung von EX1 (Ersetzen von .PST durch .OUT) sowohl Postfixnotation als auch sequentieller Code produziert werden. Ferner wurde durch weitgehende Erhaltung der urspruenglichen Meta-Beschreibung und deren Notationsart auch deren Klarheit und Uebersichtlichkeit erhalten. Explizit bedeutet das, dass die fuer (D6) und (D7) benoetigten Funktionen nur in der erweiterten Meta-Beschreibung

enthalten sein muessen und in Faellen wo dies notwendig scheint, immer noch von derselben urspruenglichen minimalen Selbstbeschreibungsversion (A-6-1) ausgegangen werden kann. Dies trifft insbesondere fuer Implementierungen zu, die entweder keine Infixterme verarbeiten (Abschnitt 3.2.3) oder Postfixnotation produzieren muessen. Ueberdies kann dadurch eine erhebliche Erweiterung des urspruenglichen META-II/X-Systems erreicht werden, ohne das System neu zu konzipieren, wie dies bei Meta-3 (8) und Meta5 (9) geschah.

2.3.3 Symbolprozessor

Die Verwendbarkeit des META-II/X-Systems als Symbolprozessor ist dadurch gegeben, dass die Meta-Maschine die Moeglichkeit besitzt, symbolischen Output zu produzieren. In der gegenwaertigen Form (A-7) ist es zwar prinzipiell – ebenso wie Meta5 – geeignet, z.B. 'Source- to source-language'-Uebersetzungen durchzufuehren; es muessten aber noch die zur praktischen Realisierung notwendigen Funktionen bereitgestellt werden. Dies trifft insbesondere auf die fuer eine Uebersetzung in Assembler sich als geeignet erwiesenen Output-Befehle mit starrem Outputformat zu. Ein in der bestehenden Version (A-7) moeglicher Einsatz als Symbolprozessor waere z.B. bei der symbolischen Differentiation von mathematischen Ausdruecken denkbar.

Eine solche Aufgabe laesst sich durch META-II/X mit relativ wenig Aufwand realisieren. Eine Differentiation des Ausdrucks

(D8) y = d/dx(a0+a1*x+a2*x**2+...+an*x**n)

laesst sich z.B. mit folgender Formulierung erreichen:

```
(D9)   EX0 ::= 'y' '=' 'd/dx' '(' EX1 ')' ;
       EX1 ::= EX2 $ ('+' EX2 / '-' EX2) ;
       EX2 ::= .NUM .OUT(*) '*' EX3 ;
       EX3 ::= 'x' .OUT(*) '**' .NUM .OUT('***' * '-1') / .EMPTY ;
```

3 CAMAC IML PRECOMPILER

Der mit dem META-II/X-System produzierte IML-Compiler (A-6-3) basiert auf dem in 2.3.2 dargestellten Compilerzustand. In der speziellen Implementierung ergibt sich, dass die Form der Syntaxgleichungen sehr stark von den darin eingebetteten semantischen Outputbefehlen beeinflusst werden. Im Normalfall gilt, dass der einem erkannten syntaktischen Element der Sourcesprache entsprechende Objectcode ausgegeben werden muss, bevor der naechste Ausdruck im Input untersucht wird. Ausnahmen hierzu bilden die Backup- und die Universalregister-Instruktionen (vgl. A-2). Werden diese Ausnahmen in konkreten Implementierungen nach Moeglichkeit vermieden – was insbesondere bei einfachen Syntaxstrukturen mceglich ist – , so koennen mit META-II/X sehr kurze und effektive Compiler formuliert werden.

Eine Hauptbedingung bei der Entwicklung von IML durch die ESONE-SWG war der Wunsch nach einer einfachen Syntaxstruktur, um erstens eine leicht

verstaendliche und leicht erlernbare Sprache zu haben und zweitens die Implementierung so einfach wie moeglich zu halten. Neben anderen IML-Implementierungen (Macro-Assembler-Implementierung, Zusammenstellung siehe (12,6)) haben hierzu auch erste Testversionen des vorliegenden IML-Compilers (A-6-3) beigetragen (13,5).

Da ein mit META-II/X erstellter Compiler in jedem Fall symbolischen Output produziert, ergibt sich hieraus, dass der bei einer Uebersetzung zu bewaeltigende Aufwand eine unmittelbare Funktion des Levelunterschiedes zwischen Source- und Object-Sprache bzw. der durch die Object-Sprache darstellbaren Semantik ist (14). Aus diesem Grund wird im weiteren Verlauf ausfuehrlich auf die syntaktischen und semantischen Eigenschaften beider Sprachen eingegangen.

3.1 Source- und Object-Sprache

3.1.1 CAMAC IML

IML ist eine Sprache zur Programmierung der Kommunikation zwischen Computer und CAMAC-Peripherie in einem CAMAC-Computer-System und erlaubt die rechnerunabhaengige Darstellung von CAMAC-Peripheriefunktionen. IML ist so definiert (6), dass sie den niedrigsten implementierungsunabhaengigen Softwarelevel ueber den Spezifikationen der CAMAC-Hardware (15,16) besitzt.

Sie beschreibt eine Serie von aufeinanderfolgenden Hardwarelevels, angefangen von einem einfachen Register in einem CAMAC-Modul bis hin zur vollstaendigen CAMAC-Peripheriestruktur. Ein Register wird in IML-Deklarationsstatements durch eine Liste der Systemknotenpunkte beschrieben. Die Liste der Knotenpunkte repraesentiert ein 4-tupel von Werten fuer die Adressen von Branch, Crate, N-Station und A-Substation. Die Systemstruktur der CAMAC-Peripherie ist eine lineare Baumstruktur und wird in IML implizit durch die Gesamtheit ihrer Knotenpunkte dargestellt (12). Die IML Aktionsstatements beschreiben die von der CAMAC-Hardware vorgegebenen I/O-Modi (15,16).

In syntaktischer Hinsicht stellt IML eine Sprache mit einer starren, einfachen Syntaxstruktur dar. Die typische Form z.B. eines Aktionsstatements ist gegeben durch:

(D6) <mode> <function> <ext> <int>

mit

<mode>	=	I/O-Modus
<function>	=	Mnemonic fuer CAMAC-F-Code, <F>
<ext>	=	CAMAC-Adresse (BCNA), die <B>,<C>,<N>,<A> repraesentiert
<B>	=	Branch
<C>	=	Crate
<N>	=	N-Station
<A>	=	A-Substation
<F>	=	F-Code
<int>	=	Interne Referenz (Kernspeicheradresse)

Sie ist zwar hardwareorientierte Sprache, besitzt aber aufgrund ihrer

Kompaktheit hinsichtlich der Deklaration der CAMAC-Peripherie und der Beschreibung der I/O-Modi in den Aktionsinstrunktionen charakteristische Zuege einer Intermediate-Level-Sprache.

Da die IML Spezifikation (6) direkt auf der CAMAC-Hardware-Spezifikation aufbaut, ist die Semantik von IML unmittelbar auf die CAMAC-Systemstruktur und - wegen der Wahrung der Rechnerunabhaengigkeit - auf den CAMAC-spezifischen I/O-Mechanismus zugeschnitten. Die in (D6) aufgefuehrte Syntax entspricht jedoch mehr der vom Programmierer gewuenschten Form. Hinsichtlich der Informationsverarbeitung beim Parsing eines solchen Statements besteht ein Teilproblem im wesentlichen darin, CAMAC-Worte (BCNAF's) der Form:

(D7) <B><C><N><A><F>

zu produzieren. Weiterhin ergeben sich je nach Rechnertyp spezifische Uebersetzungsprobleme, die vom verschiedenen I/C-Handling herruehren, da die heute existierenden Rechner i.a. von der CAMAC-Hardware abweichende Peripherie-Strukturen und I/C-Mechanismen besitzen.

3.1.2 COMPASS Assembler

Der von dem in A-6-3 aufgefuehrten IML-Compiler produzierte Objectcode ist die COMPASS Assemblersprache (COMPASS) der Rechnerserie CDC 3L00 (7). COMPASS ist eine Ein-Adress-Assemblersprache mit starrem Format und besitzt ein grosses Befehlsrepertoire an ausfuehrbaren und deklarativen Instruktionen. Die Syntax einer COMPASS-Instruktion ist gegeben durch:

(E3) (<label>) <operator> <operand>

Eine Liste aller verwendeten Instruktionen und deren Bedeutung kann der Referenz (7) entnommen werden. Da eine detaillierte Diskussion des IML-Compilers in Abschnitt 3.2.3 stattfindet, sollen im folgenden nur diejenigen Eigenschaften von COMPASS explizit aufgefuehrt werden, die wesentlich zu einer Vereinfachung der Uebersetzung und damit auch der Compilerbeschreibung beigetragen haben, naemlich

(E4) Arithmetische Ausdruecke als Operand,
Literaldefinition der Operanden,
Variable Felddefinition eines Wortes.

Diese Eigenschaften sind ebenfalls bei der Frage nach der formellen Uebertragbarkeit des vorliegenden IML-Compilerkonzeptes fuer andere Zielrechner von entscheidender Bedeutung.

Die konkrete Ersparnis an Uebersetzungsaufwand durch obige Eigenschaften (E4) sei anhand der Instruktion 'VFD' (variable Felddefinition) erlaeutert. Die in dem IML-Read-Statement

(D8) SA READ ADC VALUE

verwendete externe CAMAC-Adresse ADC, die durch das IML Declaration Statement

(D9) LOCD ADC H 1,2,3,4

deklariert werden kann, muss bei der Uebersetzung von IML ein CAMAC-Wort reservieren, das mit den entsprechenden Werten von Branch, Crate, N-Station und A-Substation initialisiert wird. Mit der Moeglichkeit der variablen Felddefinition kann die entsprechende 'VFD'-Instruktion

(E7) VFD 06/1,04/2,05/3,05/4

aus (D9) unmittelbar abgeleitet werden.

3.2 Generierungsverfahren

Um einen Compiler mit META-II/X zu erstellen, muss zunaechst untersucht werden, ob die in SIM zur Verfuegung stehenden Funktionen ausreichen, um die gewuenschte Uebersetzung der gegebenen Sprache - in diesem Fall, IML - durchfuehren zu koennen. Geht man von der Selbstbeschreibungsversion (A-6-1) bzw. den dazu notwendigen Simulatorfunktionen aus, so wird die Menge der fuer die Selbstbeschreibung von CMCO benoetigten Funktionen i.a. nicht ausreichen. Nach erfolgter Spezifikation der zusaetzlich benoetigten Simulatorfunktionen kann die Implementierung eines Precompilers in zwei Schritten, dem Expansionszyklus und der Precompilerdefinition, vorgenommen werden.

In den folgenden drei Abschnitten werden die fuer die Implementierung eines IML-Precompilers notwendigen Erweiterungen von SIM, der Expansionszyklus und die Compilerdefinition diskutiert. Das hier aufgezeigte Verfahren ist nicht nur auf IML beschraenkt, sondern kann prinzipiell auf jede mit META-II/X zu uebersetzende Sprache angewandt werden.

3.2.1 Simulatorfunktionen

Wie leicht einzusehen ist, fuehrt die Problemanalyse der Uebersetzung von IML in COMPASS auf folgende zu realisierende Funktionen:

1. Generierung von mehreren voneinander unabhaengigen Systemlabeln.

2. Uebergabe von Systemlabeln an andere Produktionsregeln durch stackfreie Register. (Dies ist unmittelbar aus der rekursionsfreien Definition des IML-Compilers ableitbar (siehe Abschnitt 3.2.3, sowie A-6-3).)

3. Kopieren von Strings aus dem Input in den Output (Aktualparameter des Objectcodes).

4. Kopieren von compilerinterner Information in den Output (Objectcode).

5. Outputformattierung und Diagnostic-Outputfunktionen.

6. Tests fuer Identifier, Strings und Zahlen.

7. Erkennen von Host-Language-Statements.

8. Erkennen von Syntaxfehlern mit anschliessender Weiterfuehrung des Parsings.

Diese in der IML-Compilerbeschreibung benoetigten Funktionen (Metafunktionen) muessen zunaechst in Termen von Simulatorfunktionen (Grundfunktionen) definiert werden. Die Anzahl der benoetigten Grundfunktionen haengt von der Anzahl und der Verschiedenartigkeit der Metafunktionen ab. Durch geschickte Wahl der einzelnen Simulatorfunktionen laesst sich eine minimale Menge von Grundfunktionen finden, die alle benoetigten Metafunktionen bilden koennen. Eine minimale Menge von Grundfunktionen fuehrt jedoch auf entsprechend groessere interne Compilerdarstellungen und ist in den meisten Faellen unerwuenscht. Das Optimum zwischen der Menge der Metafunktionen und der Menge der Grundfunktionen bzw. der Menge der eine Metafunktion aufbauenden Grundfunktionen kann nicht generell gefunden werden, sondern ergibt sich durch die im Einzelfall vertretbaren Groessen der internen Compilerdarstellungen (siehe Abschnitt 3.2.3).

In den meisten Faellen sind die Grundfunktionen so gewaehlt, dass eine Metafunktion nur aus der Kombination von zwei Grundfunktionen besteht. Als Beispiel hierfuer sei die Metafunktion (Testfunktion) '.ID' (A-6-1) angegeben, die z.B. in die Folge 'ID', 'BF A11', von Grundfunktionen uebersetzt wird (vgl A-3, LN53-54). Ein Beispiel fuer eine Metafunktion, die aus mehr als zwei Grundfunktionen besteht, ist z.B. durch die Outputformatfunktion '.OUT(-CLL- *)' gegeben, die durch die Folge 'CL -CLL-', 'CI', 'OUT' von Grundfunktionen (vgl. A-3, LN55-57) dargestellt wird.

Geht man von der in A-6-1 aufgefuehrten Beschreibung CMS0 aus, so betreffen die im vorliegenden Fall der IML-Uebersetzung zu realisierenden Simulatorfunktionen bzw. Erweiterungen von vorhandenen Grundfunktionen

- Generierung und Uebergabe von Systemlabeln (1.,2.)

- Outputformattierung (5.)

- Erkennen von Host-Language-Statements und Syntaxfehlern (7.,8.)

Die Spezifikation dieser Grundfunktionen kann A-2 entnommen werden. Nach Eingliederung der fuer die IML-Uebersetzung benoetigten Grundfunktionen in SIM und ASS kann - wie im folgenden Abschnitt 3.2.2 ausgefuehrt wird - die Erweiterung der Metabeschreibung erfolgen.

3.2.2 Erweiterungszyklus

Die Erweiterung der Metabeschreibung CMS0 geschieht nach dem in 2.1.2 erlaeuterten Verfahren (Abb. 2). Sie dient in erster Linie dazu, die fuer eine spaetere Uebersetzung benoetigten Grundfunktionen bereitzustellen. Hierzu ist notwendig, dass ein erweiterter Compiler CMC1 durch seine entsprechende Metabeschreibung CMS1 definiert wird (Anhang A-6-2). Weiterhin kann durch geeignete Formulierung in CMS1 die fuer die Erstellung des Compilers CMS(Y) zu verwendende Metasprache den

Erfordernissen der Sprache Y weitgehend angepasst werden, d.h. eine Aenderung der Metasprache erreicht werden. Die speziell fuer eine IML-Uebersetzung vorgenommenen Aenderungen und Erweiterungen von CMS0, die Compilerbeschreibung CMS1 (A-6-2), sei im folgenden kurz diskutiert:

1. Regel ST: ST laesst gegenueber CMS0 (A-6-1) zu, dass Produktionsregeln auch durch ';' statt '.,' abgeschlossen werden.

2. Regel EX3: .LID definiert einen Ausdruck der Form, <IDENTIFIER> <STRING>. .COM laesst Kommentare der Form, <STRING1> <BELIEBIGE STRINGFOLGE> <STRING1>, zu. .HOST gibt an, dass ein Host-Language-Statement durch die in Klammer stehenden Argumente gekennzeichnet sind. .ERP dient zur Definition eines Markierungspunktes waehrend des Parsings eines Programmes, an den im Falle eines Syntaxfehlers zurueckgesprungen wird, sodass die Uebersetzung sinnvoll weitergefuehrt werden kann.

3. Regel CUT: Die Ausgabe einer Zeile kann auch durch '$ OU1' erfolgen. (In dieser Schreibweise ist $ OU1 so zu verstehen, dass hierfuer mehrere der in Regel OU1 aufgefuehrten Alternativen stehen koennen.) Die Ausgabe einer Zeile mit Label kann auch durch '$ OU1' erfolgen. Das Uebertragen einer Inputzeile in eine Outputzeile kann durch .COPYL veranlasst werden. Ein String kann durch .OCD($ OU1) oder <$ OU1> in einen Output-Puffer uebertragen werden. .LBL OU1 oder :OU1 ermoeglichen das Uebertragen eines Labels in das Labelfeld des Output-Puffers. Speziell wird z.B. durch :*S1 der Zeichenzaehler des Output-Puffers auf eins gesetzt, ohne dass ein Zeichen uebertragen wird. Durch .EOL oder .. wird der Outputpuffer abgeschlossen und eine Outputzeile ausgegeben.

4. Regel OU1: *S1, *S2, ... *C3 ermoeglichen die Zwischenspeicherung und die Ausgabe von Inputstrings in bzw. von den entsprechenden Universalregistern. *1, *2, *3, *4 generiert voneinander unabhaengige Label in den bezeichneten Label-Boxen.

Mit diesen Erweiterungen kann die in A-6-2 dargestellte Compilerbeschreibung CMS1 benutzt werden, die in A-6-3 aufgefuehrte Metabeschreibung CMS(IML) eines IML-Compilers CMC(IML) zu uebersetzen (Abb. 2).

3.2.3 Compilerdefinition

CMS(IML) (A-6-3) basiert auf CMS1 (A-6-2), d.h. sie benutzt nur Metasprachelemente, die in CMS1 definiert wurden. CMS(IML) beschreibt ein IML-Subset (Abschnitt 4.1), das in COMPASS eingebettet ist. Aufgrund der Syntaxstruktur von IML konnte CMS(IML) backup- und rekursionsfrei definiert werden. Dadurch ergibt sich eine direkte Kontrolle in den einzelnen Produktionsregeln.

Ein IML-COMPASS-Programm besteht nach CMS(IML) aus einer Reihe von Statements (Regel IML und PROG). STMT beschreibt die einzelnen Statementgruppen, die durch LID ACTS, ACTS, DECL und .COM beschrieben werden. .HOST(-9 * -) charakterisiert ein COMPASS-Statement (Host-Language), das durch ein Blank in Spalte 9 oder '*' in Spalte 1 gegeben ist. LID ACTS analysiert IML-Action-Statements mit Label der

Form '<IDENTIFIER>:' (.LID(-:-)) und ACTS IML-Action-Statements ohne Label. DECL beschreibt die verschiedenen Arten von IML-Declaration-Statements. Alle IML-Statements enden mit ';'. .COM(-/*-) erlaubt das Einfuegen von Kommentaren der Form, /* <stringfolge> /*. Die aus den IML-Function-Codes bzw. Keywords zu produzierenden Octalzahlen werden in der Compilerbeschreibung in den entsprechenden Regeln definiert. Dasselbe gilt fuer die CAMAC-Adress-Deklarationen (fuer die Werte von B,C,N,A), deren Umwandlung von Dezimalzahlen zu ihren octalen Aequivalenten in den Regeln NUM1, NUMA, ... NUM6 in der Darstellung als lineare Tabelle geschieht. Von einer weiteren detaillierten IML-Compilerbeschreibung wird im folgenden abgesehen. Stattdessen werden nur die wesentlichen Einzelheiten und Besonderheiten der Compilerbeschreibung erwaehnt.

Die Objectcodeoptimierung wurde durch Verwendung von Unterprogrammen, die in Abschnitt 4.1 naeher erlaeutert werden, erreicht. Um die interne Compilerdarstellung CMC(IML) so klein wie moeglich zu halten (Compileroptimierung), wurden die Subregeln einer Produktion so formuliert, dass verschiedene Kombinationen dieser Subregeln eine uebergeordnete Regel bilden und verschiedene Kombinationen dieser Regeln wiederum eine naechst hoeheren Produktionsregel. Eine solche tiefgreifende Verflechtung der Produktionsregeln untereinander bringt erhebliche Kernspeichereinsparungen fuer CMC(IML). Bei der vorliegenden Implementierung wurde durch geschickte Codierung eine betraechtliche Reduzierung von CMC(IML) erreicht, ohne dass die Uebersetzungszeit merklich anstieg. Der Grund hierfuer liegt darin, dass in einem Teil der Faelle im wesentlichen wenig benutzte Produktionen zusammengefasst wurden. Eine Compileroptimierung durch Verflechtung der Produktionsregeln sollte daher insbesondere an Strukturen, die nur selten auftreten, vorgenommen werden.

Ein weiterer wichtiger Punkt ist, dass die Metafunktionen in CMS1 derart definiert werden, dass bei haeufigem Gebrauch dieser Funktionen in CMS(IML) die interne Darstellung CMC(IML) nicht unnoetig gross wird. In diesem Fall ist ueberdies auch eine Laufzeitreduzierung zu verzeichnen.

Ferner wurde eine Compileroptimierung dadurch erreicht, dass die Stringargumente in den Outputbefehlen - die relativ oft auftreten - nach Moeglichkeit zusammengefasst wurden und die Outputformattierung in einer Manipulationsroutine durchgefuehrt wurde. So bringt z.B. die unformattierte Ausgabe (E8.1) gegenueber der formattierten Ausgabe (E8.2)

(E8)

1

```
.OUT(-LDA- *)
CL  4
    -LDA -
CI
OUT
```

2

```
.OUT(-LDA      - *)
CL  10
    -LDA -
    -   -
    -   -
CI
OUT
```

eine Kernspeicherersparnis von 3/10, und die Zusammenfassung zweier unformattierter Ausgaben (E9.2) zu einer einzigen unformattierten Ausgabe (E9.1) eine Ersparnis von 5/10.

```
(E9)          1                    2

          <-1234->             <-12-> <-34->
           CL  4                CL  2
               -1234-                -12  -
                                 CL  2
                                       -34  -
```

Abschliessend sei darauf hingewiesen, dass alle Fragen der hier behandelten Compileroptimierung im Einzelfall gestellt und geloest werden muessen. Als Entscheidungshilfe leistet hier das Meta-Assembler-Listing von CMA(IML) wertvolle Dienste und sollte daher unbedingt verwendet werden. Fuer die Testphase eines Compilers ist jedoch eine Compileroptimierung durch obige Methoden nicht ratsam, da durch die Verflechtung der Produktionsregeln die leichte Modifizierbarkeit der Compilerbeschreibung CMS aufgegeben wird.

4 CAMAC IML UEBERSETZUNG

Eine ausfuehrliche Darstellung des gesamtet Uebersetzungsablaufes wurde bereits in (5,12) gegeben. Zusammenfassend sei hier nur angefuehrt, dass die gesamte Uebersetzung in zwei getrennten Schritten, dem Precompiler- und dem Assemblerlauf, erfolgt. Der Precompilerlauf wird an dem Zentralrechner der GfK, einer IBM/370-168, und der Assemblerlauf an dem Zielrechner, einer CDC 3100, durchgefuehrt. Im weiteren Verlauf soll nur auf die Aspekte der Pecompilerphase eingegangen werden.

Das in 3.2 aufgezeigte Generierungsverfahren eines IML Precompilers stellt eine relativ einfache und flexible Bootstrappingmethode der automatischen Compilergenerierung dar. Fuer einfache Sprachen wie z.B. IML wird nicht nur der IML Compiler CMS(IML), sondern auch der fuer die Uebersetzung von CMS(IML) benoetigte, erweiterte Compiler CMS1 automatisch erstellt. Dies ist insofern von erheblicher Bedeutung, da in CMS1 schon auf spezifische Forderungen und Eigenschaften eingegangen werden kann, die bei der spaeteren Sprachuebersetzung auftreten werden (siehe Abschnitt 3.2.2). Damit sind die Eigenschaften von CMS(IML) voll kontrollierbar und veraendertar.

Die dieser IML Implementierung zugrundeliegende IML Syntax ist die von der ESONE-SWG vorgeschlagene Macro-Expansion-Syntax (6). Diese Syntax ist in erster Linie entwickelt worden, um die Expansionsfaehigkeit von Macro-Assemblern zur Eingliederung von IML in eine Assembler Host-Language zu nutzten. Die von dieser Macro-Syntax von einem Macro-Assemtler verlangten Eigenschaften (6) sind – wie z.B. im vorliegenden Fall des COMPASS-Assemblers – nicht immer in den Macro-Assemblern vorhanden (12). Ein Beispiel hierfuer ist, dass Macro-Assembler zwei verschiedene Symboltabellen, eine fuer Macro-Namen und eine fuer normale Symbole, fuehren muessen (6). Ausserdem lehnt sich die IML-Macro-Syntax sehr an die von Macro-Assemblern benoetigte Notation an.

Weiterhin ist eine Implementierung durch Macro-Assembler sehr stark an den verwendeten Rechnertyp bzw. an dessen Betriebssystemkomponente, die Assemble-time-Software, gebunden. Diese Kopplung ist jedoch nicht nur

auf der Ebene der zu uebersetzenden Sprache - wo es insbesondere zu wuenschen ist - vorhanden, sondern auch auf der Ebene der Uebersetzerdefinition. Die Uebersetzereigenschaften sind damit fest vorgegeben und meist nur durch Eingriffe in die vom Hersteller gelieferte Software moeglich. Eine Ausnahme hiervon bilden die sogenannten systemunabhaengigen Macro-Generatoren (Prozessoren), die in kleineren Rechnern jedoch meist nicht vorhanden sind.

Die hier dargestellte IML-Implementierung hingegen bietet gegenueber Macro-Assembler-Implementierungen in diesen Punkten die Vorteile der Rechner- und Systemunabhaengigkeit sowie die vollkommen freie Wahl der Notation auf der Ebene der Compilerdefinition. Die fuer META-II/X notwendige Voraussetzung ist, unabhaengig von der Groesse des zu uebersetzenden Programmes, lediglich die Faehigkeit, FORTRAN IV Programme ausfuehren zu koennen, und, bei Vorhandensein von Overlay-Moeglichkeiten, einen Mindestkernspeicherbedarf fuer Programme von ungefaehr 24k Bytes zur Verfuegung stellen zu koennen (vgl. A-5). Wie ein Vergleich des IML Compilers CMC(IML) mit entsprechenden Macro-Assemblern fuer den etwa gleichen Sprachumfang von IML zeigt, ist der in Macro-Assemblern (17) benoetigte Assembler-time-Kernspeicherbedarf sehr stark von der Groesse der zu uebersetzenden Programme abhaengig und groesser als der von der Meta-Maschine (SIM+CMC(IML)) benoetigte Bereich.

4.1 Subset Implementierung

4.1.1 Umfang

Das durch CMS(IML) definierte IML Subset unterscheidet sich von dem in (6) definierten IML Set im wesentlichen durch eine macrofreie Syntaxnotation. Die fuer Macro-Implementierungen notwendigen Declaration Statements konnten, soweit sie nicht fuer Speicherplatzreservierung und Initialisierung notwendig sind, entfallen. Damit ergibt sich fuer den Benutzer der Vorteil, keine Anweisungen schreiben zu muessen, die lediglich als Uebersetzungshilfe fuer den Macro-Assembler dienen. So wurden z.B. alle in der Macro-Expansion-Syntax (6) aufgefuehrten Export/Import Declarations nicht uebernommen und stattdessen in der Assembler Host-Language belassen, die ebenfalls die Moeglichkeit vorsieht, Symbole zu importieren oder zu exportieren. Es wurde hierbei die Faehigkeit des Assemblers benutzt, nichtdefinierte Symbole als fehlerhaft anzuzeigen, sodass das Halten einer Symboltabelle bei solchen Implementierungen nicht notwendig ist.

Die derzeitige Implementierung umfasst folgendes IML Subset (A-6-3):

Declarations : Hardware Address Declaration
Given Array Declaration
Lam Declaration
Channel Declaration

Actions : Single Actions (SA,SJQ,SJNQ)
Uni-Device Block Transfer (UBL)
Multi-Device Action (MA)
Lam Actions
System Actions

Die in IML enthaltenen Action Statements koennen in zwei Gruppen eingeteilt werden, die der synchronen und die der asynchronen I/O-Instruktionen. Der synchrone I/O wird ueber kurze, nichtunterbrechbare Basisroutinen (Abb. 4,18) durchgefuehrt:

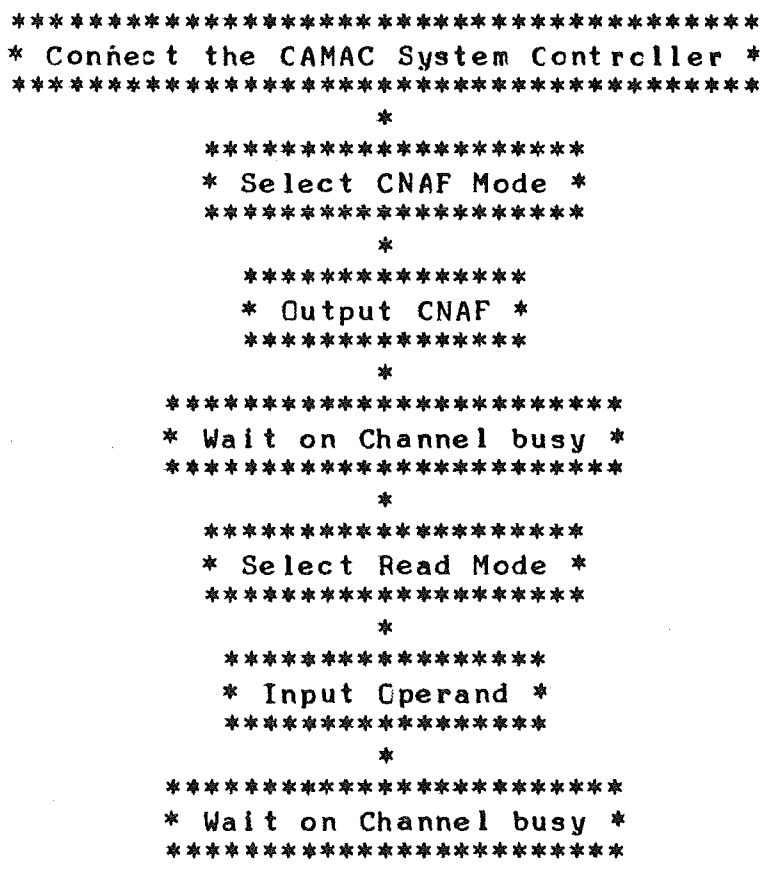


Abb. 4 I/O-Basisroutine fuer CAMAC Input-Statement

Fuer Output-Statements hat obige Routine in den entsprechenden Punkten 'Select Write Mode' und 'Output Operand'. Fuer I/C Kontrolloperationen hat die Basisroutine ebenfalls die obige Form, die Punkte 'Select Read Mode', 'Input Operand' und das zweite 'Wait on Channel busy' entfallen. Insgesamt ergeben sich aus Gruenden der Codeoptimierung 5 zentrale CAMAC-I/O-Basisroutinen fuer Input, Output und Kontroll- oder Statusinformation (18).

Asynchrone Instruktionen sind solche, die CAMAC Lam-Handling bzw. Interrupts beinhalten; sie initialisieren und fuehren Ihre I/O-Aktionen mit der CAMAC Hardware ueber die oben erwaehnten Basisroutinen aus. Die hierzu notwendige Verwaltungsarbeit wird von einem eigens hierfuer entwickelten CAMAC Lam Handler uebernommen (18).

4.1.2 Effektivitaet

Die Wahl der I/O-Basisroutinen wurde so getroffen, dass bei Verwendung dieser Routinen die Anzahl der auszufuehrenden Befehle maximal 1/10 groesser wurde, als dies bei der Ausfuehrung ohne diese Routinen der Fall waere. So ist z.B. fuer die meisten Single Actions der durch die Basisroutinen erreichte Codeoptimierungsfaktor (Verhaeltnis zwischen abzusetzendem Code und Basisroutine) ungefaehr 1:3. Das bedeutet eine Kernspeichererspamis von 75/100 gegenueber einer Laufzeitverlaengerung von nur 1/10.

Die Uebersetzung von IML in COMPASS kann als Precompilierung unter Erzeugung von Inline-Objectcode aufgefasst werden. Im Gegensatz zu 'Subroutine Calls' wurde aus Laufzeitgruenden auf das Retten der zur Ausfuehrung von IML-Instruktionen notwendigen Register verzichtet. Da die Host Sprache eine Assembler Sprache ist, koennen die hierzu notwendigen Aktionen fuer den Einzelfall dem Benutzer ueberlassen werden.

4.2 Implementierungserweiterungen

Hinsichtlich einer Implementierungserweiterung des vorliegenden IML Subsets ergeben sich dank der ausgezeichneten Erweiterungsfaehigkeiten des META-II/X Systems keine Schwierigkeiten. So wurde im vorliegenden Fall von IML zunaechst mit einem relativ einfachen, nur aus Declarations und Single Actions bestehendem IML-Subset begonnen. Nach der Erweiterung der Implementierung auf LAM Actions kann z. B. der I/O-Teil des CAMAC LAM Handlers weitgehend in IML geschrieben werden, ohne an Effektivitaet einzubuessen.

Von entscheidender Bedeutung fuer eine Implementierungserweiterung ist, wie im Falle der gesamten IML Implementierung, die verwendete Object Sprache. Insbesondere ist dies die Faehigkeit der Object Sprache, die durch IML Statements dargestellte Semantik in geeigneter Weise ausdruecken zu koennen. Eine vertretbare Laenge der internen Compilerdarstellung CMC(IML) wird nur durch eine komfortable Assemblersprache gewaehrleistet. Dadurch koennen Uebersetzungsarbeiten in die Assemblerebene verlagert werden.

5 UEBERTRAGBARKEIT

Die Uebertragbarkeit des FORTRAN Programmes wird dadurch sehr erleichtert, dass an zentraler Stelle (BLOCK DATA Unterprogramm) auf die Characterrepraesentation der IBM/370 eingegangen wird. Die hierfuer relevanten Parameter sind die Charactermaske, CHMASK, und die Bytelaenge, LMASK (vgl. A-7).

Um die Uebertragbarkeit der internen Compilerbeschreibung zu sichern, wurde die Moeglichkeit der Ein-/Ausgabe von Compiler Beschreibungen in der symbolischen Assemblersprache MA geschaffen.

LITERATURVERZEICHNIS

(1) Chomsky, N., 'Syntactic Structures', Mouton s'Gravenhage (1957)

(2) Feldmann, J., Gries, D., 'Translator Writing Systems', Comm. of the ACM,11,77 (1968)

(3) Schorre, D. V., 'META-II, a Syntax Oriented Compiler Writing Language', Proc. 19th ACM Natl. Conf.,D1 (1964)

(4) Karbstein, W., Koegel, B., 'CAMAC Multi-Branch System for CDC 3100 Computers', CAMAC Bulletin,10 (1974)

(5) Kneis, W., 'Implementation of CAMAC IML in an Assembler Language Environment', CAMAC Bulletin,10 (1974)

(6) CAMAC. The Definition of IML, ESONE Committee, ESONE/IML/01 (1974)

(7) COMPASS Ref. Man., Control Data Corporation, Ref. Nr. 60236800 (1972)

(8) Schneider, F. W., Johnson, C. D., 'Meta-3, a Syntax Directed Compiler Writing Compiler to Write Efficient Code', Proc. 19th ACM Natl. Conf.,D1.5-1 (1964)

(9) Oppenheim, D. K., Haggerty, D. P., 'Meta5: a Tool to Manipulate Strings of Data', Proc. 21th ACM Natl. Conf.,465 (1966)

(10) O'Neil, J. T., Jr., 'Meta Pi - An Interactive Cnline Compiler-Compiler', Proc. AFIPS, FJCC,33,201 (1968)

(11) Chandler, W. J., 'Meta S: A Metalanguage for Compilers', Proc. 6th Hawaii Int. Conf. On Systems Sciences (1973)

(12) Kneis, W., Degenhardt, K. H., Woletz, W., 'Die Uebersetzung der CAMAC Sprache unter Verwendung der Zwischensprache IML - Erfahrungen bei der Implementierung von CAMAC Compilern', Proc. GI, GfK, VDI/VDE-GMR Fachtagung, Prozessrechner 1974, Karlsruhe, Springer Verlag (1974)

(13) Kneis, W., 'General Aspects of Translation and Implementation of the CAMAC IML', ESONE-Software-Working-Group, Draft Report, (1973)

(14) Kneis, W., 'Implementierung von CAMAC durch Compiler', Proc. First International Symposium on CAMAC in Real-Time Computer Applications, Luxembourg (1973)

(15) CAMAC. A Modular Instrumentation System for Data Handling, EURATOM Report, EUR 4100e (1972)

(16) CAMAC. Organisation of Multi Crate Systems, EURATOM Report, EUR 4600e (1972)

(17) Kubitz, M., Kind, R., 'Macro IML Implementations for PDP 11 Computers', Hahn-Meitner Institut, Berlin, Interner Bericht (in Druck)

(18) Kneis, W., Kartstein, W., Volk, B., 'CAMAC IML – Benutzeranleitung', GfK Karlsruhe (1975) unveroeffentlicht

ANHANG

A-1 Steuerkommandos des META-II/X-Systems

Syntax der Kommandosprache

Die Notation entspricht der Meta-Sprache MS des META-II/X-Systems.

```
KOMMANDO  = '$' KEY ;
KEY       = 'M' LET <PARM> / 'C' LET <PARC> /
            'A' LET <PARA> / '*' LET        /
            ' ' LET        / '$' LET  ;
PARM      = ',M' LET / ',N' LET ;
PARC      = ',I' LET / ',T' LET <SUB> / ',L' LET <SUB> /
            ',F' LET <SUB> / ',C' LET ;
PARA      = ',L' LET <SUB> / ',M' LET / ',N' LET ;
SUB       = '(' NUMBER ',' NUMBER ')' / .EMPTY ;
LET       = (LETTER/DIGIT) ... / .EMPTY ;
NUMBER    = DIGIT ... ;
```

Im Gegensatz zu MS bedeuten die Symbole '<'und '>', dass der umschlossene Parameter optional ist.

Semantik der Kommandosprache

Kommandoname:	Key:	Bedeutung:
META	M	Dieses Kommando muss immer vorhanden sein. Es wird benoetigt, um die interne Darstellung des Compilers in die von SIM oder ASS vorgesehenen Datenbereiche zu laden. Parameter: M – Laden eines CMC-Compilers, wenn nicht spezifiziert, Laden der Originalversion CMC0. N – Laden eines CMA-Compilers.
COMPILE	C	Dieses Kommando wird nicht immer benoetigt. Es veranlasst die Ausfuehrung der Uebersetzung von Meta-Beschreibungen in die spezifizierte Assemblersprache. Parameter: I – Listing des Inputs von SIM. T – Promptes SIM-Output-Listing. Bereichsangabe moeglich.

Kommandoname:	Key:	Bedeutung:
		L – SIM-Output in spezifiziertem Format (CDC-COMPASS-Assembler Format). Wird nur ausgefuehrt bei fehlerfreier Uebersetzung. Bereichsangabe durch Angabe der Zeilennummern (Default = 0-1000).
		F – Trace-Output von SIM (debug facility). Bereichsangabe durch Angabe der Zeilennummern (Default = 0-1000).
		C – SIM-Output in Karten-Format auf Plattenfile.
ASSEMBLE	A	Dieses Kommando wird nicht immer benoetigt. Es veranlasst die Ausfuehrung eines Assemblerlaufs eines CMA-Compilers. Es dient zur Herstellung von CMC-Compilern. Parameter: L – ASS-Output-Listing. Bereichswahl durch Angabe der Zeilennummern (Default =0-1000). M – CMC auf Plattenfile schreiben. N – CMA auf Plattenfile schreiben. C – frei
COMMENT	*	Kommentar-Karte
RUN		Das Run-Kommando startet unverzueglich die Ausfuehrung des bisher eingelesenen Jobs. Dem Run-Kommando folgende Karten werden in diesem Lauf nicht beruecksichtigt. Parameter: keine
END	$	Das Endkommando wirkt als EOF-Karte und beendet einen META-II/X-Lauf.

A-2 Befehlsliste von SIM und ASS

MC	Op	MA Operand	Bedeutung/Aktion
1	SET		Setze T/F-Schalter, S = w.
2	NOB		Blank ist Begrenzer von syntaktischen Einheiten.
3	RDB		Blank ist nicht Begrenzer von syntaktischen Einheiten.
4	TST	'STRING'	Teste, ob das im Input vorliegende String mit dem Operanden uebereinstimmt. Wenn ja, S=w, wenn nein, S=f.
5	ID	IDENTIFIER	Teste, ob das im Input vorliegende String ein Identifier ist. Wenn ja, S=w, wenn nein, S=f.
6	NUM	NUMBER	Teste, ob das im Input vorliegende String eine Zahl ist. Wenn ja, S=w, wenn nein, S=f.
7	SR		Teste, ob das im Input vorliegende String von der Form '....' ist. Wenn ja, S=w, wenn nein, S=f.
8	CLL	ADDRESS	Rekursivaufruf der Routine ADDRESS.
9	R		Rekursive Rueckkehr aus aufgerufener Routine.
10	BT	LABEL	Verzweige nach LABEL, wenn S=w, sonst p=p+1.
11	B	LABEL	Verzweige nach LABEL.
12	BF	LABEL	Verzweige nach LABEL, wenn S=f, sonst p=p+1.
13	BE		Fehlerdiagnostic- und Backup-Routine wenn S=w, p=p+1 wenn S=f, und up=w, backup in input und output, oder erp=w, Ausgabe einer Fehlermeldung und Rueckkehr zu einem definierten Ausgangspunkt (markiert durch ERP), oder sonst, Fehlerstop und Ausgabe einer Fehlermeldung.
14	OUT		Ausgabe einer Zeile.
15	CL	STRING	Kopiere Operand in Output-Puffer.
16	CI		Kopiere ein im Input vorliegendes und getestetes String in Output-Puffer.
17	GN	N	Generiere ein Label in Box N, N=1-4.
18	COM	STRING	Teste, ob ein Kommentar vorliegt, wenn ja, S=w, wenn nein, S=f. Ein Kommentar ist dann vorhanden, wenn eine Zeichenfolge von dem im Operandenfeld angegebenen STRING eingeschlossen wird.

MC	MA Op	MA Operand	Bedeutung/Aktion
19	LID	STRING	Teste, ob ein Ausdruck der Form Identifier, gefolgt von STRING vor liegt (z.B. STRING = :, dann ist ABC: ein Ausdruck der gewuenschten Form). Wenn ja, S=w, sonst S=f.
20	ERP		Merke aktuelle Compileradresse als Fehlerrueckkehrpunkt.
21	LB		Bereite Ausgabe eines Systemlabels vor (Spaltenzaehler in Output-Puffer auf eins setzen.).
22	MS		Ausgabe einer spezifischen Diagnostic-meldung, die als Argument spezifiziert sein muss.
23	ST	N	Kopiere das vorliegende Inputstring in das Zwischenregister N, N=1-3. Auf diese Register kann von jeder Stacktiefe aus zugegriffen werden.
24	CLS		Kopiere ein in MC vorliegendes String in das Zwischenregister N=1.
25	HOS		Teste, ob das im Input vorliegende Statement ein Host-Statement ist (Spezifikation des Host-Statements durch Argument), wenn ja, kopiere es in den Output und S=w, wenn nein, S=f.
26	CS	N	Kopiere den Inhalt des Zwischenregisters N in den Output-Puffer.
27	UP		Schalte Backup-Mode ein, UP=w.
28	CIO		Kopieren Inputzeile in Output-Puffer.
29	EOF		Teste, ob End-of-File vorliegt.
30	FOP*	STRING	Finde naechsten Operator in Stack.
31	OP *	STRING	Teste, ob naechstes String ein Operator ist.
32	CIP*		Kopiere Input in Stack.
33	CLP*		Kopiere Literal in Stack.
34	POT*		Kopiere Stack in Output und clear Stack.
35	HV *		Generiere Hilfsvariable (HV) in HV-Liste (wenn nicht vorhanden) und kopiere HV-Name in Stack und Output-Puffer.
36	HLV*		Kopiere gesamte HV-Liste in Output-Puffer zur Speicherplatzreservierung.
37			frei
38			frei
39			frei
40			frei
	ADR	ADDRESS	Pseudoinstruktion fuer Anfangsadresse eines Programmes (Compilers).
	END		Pseudoinstruktion fuer Endadresse eines Programmes (Compilers).

Testbefehle:

Alle Testbefehle (TST, ID, NUM, SR) ueberspringen saemtliche Blanks bis zu Beginn der entsprechenden Input-Zeichenkette. Nach erfolgreichem Test

ist der Wert des Inputzeichen-Zaehlers dem, der Zeichenkette folgenden Zeichen zugeordnet. Die vorhergehende Position des Inputzeichen-Zaehlers ist noch bekannt.

Rekursivbefehle:

Die Uebergabe der Parameter zwischen CLL und R geschieht ueber eine Push-Down-Stack-Tabelle.

Fuer IML-Uebersetzung benoetigte oder modifizierte Befehle:

13, 17-20, 22-28

Die mit * gekennzeichneten Befehle sind in der vorliegenden META-II/X Version (A-7) nicht implementiert.

A-3 Simulator-Listing von CMS0 (A-6-1)

```
*************************************************************************
     Zeile                  Objectcode                  Sourcecode
*************************************************************************
    LN    1 (      7)        ADR PRO                     PRO
    LN    2 (     13) OU1                               CU1
    LN    3 (     26)        TST x*1x                    x*1x
    LN    4 (     32)        BF  A01                     x*1x
    LN    5 (     81)        CL  xGN x                   xGN x
    LN    6 (     98)        CL  x1x                     x1x)
    LN    7 (    122)        OUT                         )
    LN    8 (    167) A01                               /
    LN    9 (    174)        BT  A02                     /
    LN   10 (    184)        TST x*x                     x*x
    LN   11 (    190)        BF  A03                     x*x
    LN   12 (    239)        CL  xCI x                   xCI x)
    LN   13 (    263)        OUT                         )
    LN   14 (    308) A03                               /
    LN   15 (    316)        BT  A02                     /
    LN   16 (    334)        SR                          .STRING
    LN   17 (    340)        BF  A04                     .STRING
    LN   18 (    389)        CL  xCL x                   xCL x
    LN   19 (    402)        CI                          *)
    LN   20 (    427)        OUT                         ).
    LN   21 (    472) A04                               .
    LN   22 (    483) A02                               .
    LN   23 (    489)        R                           .,
    LN   24 (    497) CUT                               CUT
    LN   25 (    530)        TST x.OUTx                  x.OUTx
    LN   26 (    536)        BF  A05                     x.CUTx
    LN   27 (    547)        TST x(x                     x(x
    LN   28 (    552)        BE                          x(x
    LN   29 (    581) A06                               $
    LN   30 (    587)        CLL OU1                     OU1
    LN   31 (    593)        BT  A06                     CU1
    LN   32 (    595)        SET                         OU1
    LN   33 (    599)        BE                          CU1
    LN   34 (    610)        TST x)x                     x)x
    LN   35 (    615)        BE                          x)x
    LN   36 (    659) A05                               /
    LN   37 (    666)        BT  A07                     /
    LN   38 (    676)        TST x.LABELx                x.LABELx
    LN   39 (    682)        BF  A08                     x.LABELx
    LN   40 (    731)        CL  xLB x                   xLB x)
    LN   41 (    755)        OUT                         )
    LN   42 (    764)        CLL OU1                     OU1)
    LN   43 (    769)        BE                          OU1)
    LN   44 (    813) A08                               )
    LN   45 (    824) A07                               )
    LN   46 (    834)        BF  A09                     )
    LN   47 (    883)        CL  xCUTx                   xOUTx)
    LN   48 (    907)        OUT                         ).
    LN   49 (    952) A09                               .
    LN   50 (    962) A10                               .
    LN   51 (    968)        R                           .,
*************************************************************************
```

```
*************************************************************************
     Zeile                  Objectcode                  Sourcecode
*************************************************************************
    LN  52 (    976) EX3                               EX3
    LN  53 (    991)       ID                          .ID
    LN  54 (    997)       BF   A11                    .ID
    LN  55 (   1046)       CL   xCLLx                  xCLLx
    LN  56 (   1059)       CI                          *)
    LN  57 (   1084)       OUT                         )
    LN  58 (   1129) A11                               /
    LN  59 (   1136)       BT   A12                    /
    LN  60 (   1154)       SR                          .STRING
    LN  61 (   1160)       BF   A13                    .STRING
    LN  62 (   1209)       CL   xTSTx                  xTSTx
    LN  63 (   1222)       CI                          *)
    LN  64 (   1247)       OUT                         )
    LN  65 (   1292) A13                               /
    LN  66 (   1300)       BT   A12                    /
    LN  67 (   1310)       TST  x.IDx                  x.IDx
    LN  68 (   1316)       BF   A14                    x.IDx
    LN  69 (   1365)       CL   xID x                  xID x)
    LN  70 (   1389)       OUT                         )
    LN  71 (   1434) A14                               /
    LN  72 (   1442)       BT   A12                    /
    LN  73 (   1452)       TST  x.STRINGx              x.STRINGx
    LN  74 (   1458)       BF   A15                    x.STRINGx
    LN  75 (   1507)       CL   xSR x                  xSR x)
    LN  76 (   1531)       OUT                         )
    LN  77 (   1576) A15                               /
    LN  78 (   1584)       BT   A12                    /
    LN  79 (   1594)       TST  x(x                    x(x
    LN  80 (   1600)       BF   A16                    x(x
    LN  81 (   1608)       CLL  EX1                    EX1
    LN  82 (   1613)       BE                          EX1
    LN  83 (   1624)       TST  x)x                    x)x
    LN  84 (   1629)       BE                          x)x
    LN  85 (   1673) A16                               /
    LN  86 (   1681)       BT   A12                    /
    LN  87 (   1691)       TST  x$x                    x$x
    LN  88 (   1697)       BF   A17                    x$x
    LN  89 (   1734)       LB                          .LABEL
    LN  90 (   1740)       GN   1                      *1
    LN  91 (   1746)       OUT                         *1
    LN  92 (   1755)       CLL  EX3                    EX3
    LN  93 (   1760)       BE                          EX3
    LN  94 (   1809)       CL   xBT x                  xBT x
    LN  95 (   1817)       GN   1                      *1)
    LN  96 (   1842)       OUT                         )
    LN  97 (   1892)       CL   xSETx                  xSETx)
    LN  98 (   1916)       OUT                         ).
    LN  99 (   1961) A17                               .
    LN 100 (   1972) A12                               .
    LN 101 (   1978)       R                           .,
*************************************************************************
```

```
*************************************************************************
     Zeile                    Objectcode                  Sourcecode
*************************************************************************
    LN 102 (   1986) EX2                                  EX2
    LN 103 (   2016)         CLL EX3                      EX3
    LN 104 (   2022)         BF  A18                      EX3
    LN 105 (   2071)         CL  xBF x                    xBF x
    LN 106 (   2079)         GN  1                        *1)
    LN 107 (   2104)         OUT                          )
    LN 108 (   2149) A18                                  /
    LN 109 (   2156)         BT  A19                      /
    LN 110 (   2163)         CLL OUT                      OUT)
    LN 111 (   2169)         BF  A20                      OUT)
    LN 112 (   2213) A20                                  )
    LN 113 (   2224) A19                                  )
    LN 114 (   2234)         BF  A21                      )
    LN 115 (   2263) A22                                  $(
    LN 116 (   2289)         CLL EX3                      EX3
    LN 117 (   2295)         BF  A23                      EX3
    LN 118 (   2344)         CL  xBE x                    xBE x)
    LN 119 (   2368)         OUT                          )
    LN 120 (   2413) A23                                  /
    LN 121 (   2420)         BT  A24                      /
    LN 122 (   2427)         CLL OUT                      OUT)
    LN 123 (   2433)         BF  A25                      OUT)
    LN 124 (   2477) A25                                  )
    LN 125 (   2488) A24                                  )
    LN 126 (   2498)         BT  A22                      )
    LN 127 (   2500)         SET                          )
    LN 128 (   2504)         BE                           )
    LN 129 (   2541)         LB                           .LABEL
    LN 130 (   2547)         GN  1                        *1.
    LN 131 (   2553)         OUT                          *1.
    LN 132 (   2598) A21                                  .
    LN 133 (   2608) A26                                  .
    LN 134 (   2614)         R                            .,
    LN 135 (   2622) EX1                                  EX1
    LN 136 (   2632)         CLL EX2                      EX2
    LN 137 (   2638)         BF  A27                      EX2
    LN 138 (   2667) A28                                  $(
    LN 139 (   2696)         TST x/x                      x/x.
    LN 140 (   2702)         BF  A29                      x/x.
    LN 141 (   2751)         CL  xBT x                    xBT x
    LN 142 (   2759)         GN  1                        *1)
    LN 143 (   2784)         OUT                          )
    LN 144 (   2793)         CLL EX2                      EX2)
    LN 145 (   2798)         BE                           EX2)
    LN 146 (   2842) A29                                  )
    LN 147 (   2852) A30                                  )
    LN 148 (   2862)         BT  A28                      )
    LN 149 (   2864)         SET                          )
    LN 150 (   2868)         BE                           )
    LN 151 (   2905)         LB                           .LABEL
    LN 152 (   2911)         GN  1                        *1.
    LN 153 (   2917)         OUT                          *1.
    LN 154 (   2962) A27                                  .
    LN 155 (   2972) A31                                  .
    LN 156 (   2978)         R                            .,
*************************************************************************
```

```
*************************************************************************
     Zeile                 Objectcode                  Sourcecode
*************************************************************************
    LN 157 (   2986 ) ST                              ST
    LN 158 (   3001)        ID                        .ID
    LN 159 (   3007)        BF  A32                   .ID
    LN 160 (   3044 )       LB                        .LABEL
    LN 161 (   3055)        CI                        *
    LN 162 (   3061 )       OUT                       *
    LN 163 (   3073)        TST x=x                   x=x
    LN 164 (   3078 )       BE                        x=x
    LN 165 (   3086)        CLL EX1                   EX1
    LN 166 (   3091 )       BE                        EX1
    LN 167 (   3102)        TST x.,x                  x.,x
    LN 168 (   3107)        BE                        x.,x
    LN 169 (   3156 )       CL  xR  x                 xR  x )
    LN 170 (   3180)        OUT                       ).
    LN 171 (   3225 ) A32                             .
    LN 172 (   3235) A33                              .
    LN 173 (   3241 )       R                         .,
    LN 174 (   3249) PRO                              PRO
    LN 175 (   3262 )       TST x.SYNTAXx             x.SYNTAXx
    LN 176 (   3268)        BF  A34                   x.SYNTAXx
    LN 177 (   3281)        ID                        .ID
    LN 178 (   3286 )       BE                        .ID
    LN 179 (   3335)        CL  xADRx                 xADRx
    LN 180 (   3348 )       CI                        *)
    LN 181 (   3373)        OUT                       )
    LN 182 (   3403 ) A35                             $
    LN 183 (   3409)        CLL ST                    ST
    LN 184 (   3415 )       BT  A35                   ST
    LN 185 (   3417)        SET                       ST
    LN 186 (   3421 )       BE                        ST
    LN 187 (   3432 )       TST x.ENDx                x.ENDx
    LN 188 (   3437)        BE                        x.ENDx
    LN 189 (   3486 )       CL  xENDx                 xENDx )
    LN 190 (   3510)        OUT                       ).
    LN 191 (   3555 ) A34                             .
    LN 192 (   3565) A36                              .
    LN 193 (   3571 )       R                         .,
    LN 194 (   3584)        END                       .END
*************************************************************************
```

A-4 Assembler-Listing von CMS0 (A-6-1)

```
*************************************************************************
     Zeile        Objectcode            Sourcecode
                  Op  Operand
*************************************************************************
    LN      1     0  167                    ADR PRO
    LN      2     4    2               OU1  TST x*1x
    LN      3           x*1   x
    LN      4    12   10                    BF  A01
    LN      5    15    3                    CL  xGN x
    LN      6           xGN   x
    LN      7    15    1                    CL  x1x
    LN      8           x1    x
    LN      9    14                         OUT
    LN     10    10   24               A01  BT  A02
    LN     11     4    1                    TST x*x
    LN     12           x*    x
    LN     13    12   17                    BF  A03
    LN     14    15    3                    CL  xCI x
    LN     15           xCI   x
    LN     16    14                         OUT
    LN     17    10   24               A03  BT  A02
    LN     18     7                         SR
    LN     19    12   24                    BF  A04
    LN     20    15    3                    CL  xCL x
    LN     21           xCL   x
    LN     22    16                         CI
    LN     23    14                         OUT
                                        A04
    LN     24     9                    A02  R
    LN     25     4    4               OUT  TST x.OUTx
    LN     26           x.OUTx
    LN     27    12   38                    BF  A05
    LN     28     4    1                    TST x(x
    LN     29           x(    x
    LN     30    13                         BE
    LN     31     8    2               A06  CLL OU1
    LN     32    10   31                    BT  A06
    LN     33     1                         SET
    LN     34    13                         BE
    LN     35     4    1                    TST x)x
    LN     36           x)    x
    LN     37    13                         BE
    LN     38    10   48               A05  BT  A07
    LN     39     4    6                    TST x.LABELx
    LN     40           x.LABx
    LN     41           xEL   x
    LN     42    12   48                    BF  A08
    LN     43    15    3                    CL  xLB x
    LN     44           xLB   x
    LN     45    14                         OUT
    LN     46     8    2                    CLL OU1
    LN     47    13                         BE
                                        A08
    LN     48    12   52               A07  BF  A09
*************************************************************************
```

```
*************************************************************************
     Zeile       Objectcode              Sourcecode
                 Cp  Operand
*************************************************************************
    LN     49    15   3                    CL   xCUTx
    LN     50           xOUT x
    LN     51    14                        OUT
                                     A09
    LN     52     9                  A10 R
    LN     53     5                  EX3 ID
    LN     54    12  59                    BF   A11
    LN     55    15   3                    CL   xCLLx
    LN     56           xCLL x
    LN     57    16                        CI
    LN     58    14                        OUT
    LN     59    10 106             A11 BT   A12
    LN     60     7                        SR
    LN     61    12  66                    BF   A13
    LN     62    15   3                    CL   xTSTx
    LN     63           xTST x
    LN     64    16                        CI
    LN     65    14                        OUT
    LN     66    10 106             A13 BT   A12
    LN     67     4   3                    TST  x.IDx
    LN     68           x.ID x
    LN     69    12  73                    BF   A14
    LN     70    15   3                    CL   xID x
    LN     71           xID  x
    LN     72    14                        OUT
    LN     73    10 106             A14 BT   A12
    LN     74     4   7                    TST  x.STRINGx
    LN     75           x.STRx
    LN     76           xING x
    LN     77    12  81                    BF   A15
    LN     78    15   3                    CL   xSR x
    LN     79           xSR  x
    LN     80    14                        OUT
    LN     81    10 106             A15 BT   A12
    LN     82     4   1                    TST  x( x
    LN     83           x(   x
    LN     84    12  90                    BF   A16
    LN     85     8 132                    CLL  EX1
    LN     86    13                        BE
    LN     87     4   1                    TST  x )x
    LN     88           x)   x
    LN     89    13                        BE
    LN     90    10 106             A16 BT   A12
    LN     91     4   1                    TST  x$x
    LN     92           x$   x
    LN     93    12 106                    BF   A17
    LN     94    21                        LB
    LN     95    17   1                    GN   1
    LN     96    14                        OUT
    LN     97     8  53                    CLL  EX3
    LN     98    13                        BE
*************************************************************************
```

```
*************************************************************************
     Zeile        Objectcode          Sourcecode
                  Op   Operand
*************************************************************************
     LN     99    15    3                 CL   xBT x
     LN    100           xBT   x
     LN    101    17    1                 GN   1
     LN    102    14                      OUT
     LN    103    15    3                 CL   xSETx
     LN    104           xSET  x
     LN    105    14                      OUT
                                    A17
     LN    106     9                A12  R
     LN    107     8   53           EX2  CLL  EX3
     LN    108    12  113                BF   A18
     LN    109    15    3                 CL   xEF x
     LN    110           xBF   x
     LN    111    17    1                 GN   1
     LN    112    14                      OUT
     LN    113    10  116           A18  BT   A19
     LN    114     8   25                CLL  OUT
     LN    115    12  116                BF   A20
                                    A20
     LN    116    12  131           A19  BF   A21
     LN    117     8   53           A22  CLL  EX3
     LN    118    12  122                BF   A23
     LN    119    15    3                 CL   xBE x
     LN    120           xBE   x
     LN    121    14                      OUT
     LN    122    10  125           A23  BT   A24
     LN    123     8   25                CLL  OUT
     LN    124    12  125                BF   A25
                                    A25
     LN    125    10  117           A24  BT   A22
     LN    126     1                      SET
     LN    127    13                      BE
     LN    128    21                      LB
     LN    129    17    1                 GN   1
     LN    130    14                      OUT
                                    A21
     LN    131     9                A26  R
     LN    132     8  107           EX1  CLL  EX2
     LN    133    12  149                BF   A27
     LN    134     4    1           A28  TST  x/x
     LN    135           x/    x
     LN    136    12  143                BF   A29
     LN    137    15    3                 CL   xBT x
     LN    138           xBT   x
     LN    139    17    1                 GN   1
     LN    140    14                      OUT
     LN    141     8  107                CLL  EX2
     LN    142    13                      BE
                                    A29
     LN    143    10  134           A30  BT   A28
     LN    144     1                      SET
     LN    145    13                      BE
*************************************************************************
```

W. Kneis, META-II/X-System

```
*************************************************************************
     Zeile       Objectcode          Sourcecode
                 Op  Operand
*************************************************************************
    LN    146    21                      LB
    LN    147    17   1                  GN  1
    LN    148    14                      OUT
                                  A27
    LN    149     9                  A31 R
    LN    150     5                  ST  ID
    LN    151    12 166                  BF  A32
    LN    152    21                      LB
    LN    153    16                      CI
    LN    154    14                      OUT
    LN    155     4   1                  TST x=x
    LN    156          x=   x
    LN    157    13                      BE
    LN    158     8 132                  CLL EX1
    LN    159    13                      BE
    LN    160     4   2                  TST x.,x
    LN    161          x.,  x
    LN    162    13                      BE
    LN    163    15   3                  CL  xR  x
    LN    164          xR   x
    LN    165    14                      OUT
                                  A32
    LN    166     9                  A33 R
    LN    167     4   7              PRO TST x.SYNTAXx
    LN    168          x.SYNx
    LN    169          xTAX x
    LN    170    12 187                  BF  A34
    LN    171     5                      ID
    LN    172    13                      BE
    LN    173    15   3                  CL  xADRx
    LN    174          xADR x
    LN    175    16                      CI
    LN    176    14                      OUT
    LN    177     8 150              A35 CLL ST
    LN    178    10 177                  BT  A35
    LN    179     1                      SET
    LN    180    13                      BE
    LN    181     4   4                  TST x.ENDx
    LN    182          x.ENDx
    LN    183    13                      BE
    LN    184    15   3                  CL  xENDx
    LN    185          xEND x
    LN    186    14                      OUT
                                  A34
    LN    187     9                  A36 R
*************************************************************************
```

A-5 META-II/X - Konfigurationswerte (fuer IBM/370-168)

(Alle Angaben fuer Programme in k Bytes)

Programmbereich:	
META-II/X System (ohne Datenbereiche)	24
Hauptprozessor	7
MAIN	
CCPARM	
NEXT	
Meta Simulator	11
SIM	
INPUT	
PRINT	
Meta Assembler	5,5
ASEMBL	
Datenbereich:	
CMC(IML)	12
CMA(IML)	30
Meta Maschine (SIM+CMC(IML))	23
Meta Asembler + CMA(IML)	35,5
Uebersetzungszeit fuer CMS(IML)	7.7 sec
Compile Step	5.0 sec
Assemble Step	1.0 sec
File handling	1.7 sec
Uebersetzungszeit fuer IML Statement	100 msec
Compile Step	90 msec
File handling	10 msec

Naeherungswerte fuer Kernspeicheranforderungen in k Bytes:

****

MS = Laenge von META-II/X = 24 k
CM = Laenge der CMC Compilerdarstellung
CA = Laenge der CMA Compilerdarstellung
LL = Laenge der Labeltabellen

Naeherungswerte:

CA = 3 * CM(neu)
LL = CM(neu)

Voraussetzung fuer obige Naehrungswerte ist, dass CM(neu) > CM(alt) gilt.

W. Kneis, META-II/X-System

1. Compile Step
****

$$K = 24 + CM(alt)$$

2. Assemble Step
****

$$K = 24 + CA + \max(CM(alt), LL) + CM(neu)$$
$$< 24 + 5*CM(neu)$$

Mit diesen Angaben fuehrt die derzeitige overlayfreie Konfiguration zu folgenden Anforderungen an den Kernspeicher in k Bytes:

IML Uebersetzung	36
Erstellung von CMC(IML)	84
Erstellung von CMC1	27
Erstellung von CMC0	25,5

A-6-1 COMPILER BESCHREIBUNG CMS0

```
$META     *** COMPILER DESCRIPTION CMS0 ***
$COMPILE,INPUT,TRACE
$ASSEMBLE,LIST
$*DESCRIPTION OF THE MINIMAL COMPILER VERSION WITH SELFDESCRIBING
$*FACILITIES. IDENTICAL WITH THE BLOCKDATA VERSION OF THE FORTRAN
$*PROGRAM.
.SYNTAX PRO
OU1 = x*1x          .OUT(xGN x x1x)      /
      x*x           .OUT(xCI x)          /
      .STRING       .OUT(xCL x *).,
OUT = (x.OUTx x(x   $ OU1  x)x           /
      x.LABELx      .OUT(xLB x)  OU1) .OUT(xOUTx).,
EX3 = .ID           .OUT(xCLLx *)        /
      .STRING       .OUT(xTSTx *)        /
      x.IDx         .OUT(xID x)          /
      x.STRINGx     .OUT(xSR x)          /
      x(x EX1 x)x                        /
      x$x .LABEL *1 EX3 .OUT(xBT x *1) .OUT(xSETx).,
EX2 = (EX3  .OUT(xBF x *1) / OUT)
      $(EX3  .OUT(xBE x)   / OUT) .LABEL *1.,
EX1 = EX2 $(x/x.OUT(xBT x *1) EX2) .LABEL *1.,
ST  = .ID .LABEL * x=x EX1 x.,x .OUT(xR  x).,
PRO = x.SYNTAXx .ID .OUT(xADRx *) $ ST x.ENDx .OUT(xENDx).,
.END
$
```

A-6-2 COMPILER BESCHREIBUNG CMS1

```
$META   *** COMPILER DESCRIPTION CMS1 ***
$COMPILE,INPUT
$ASSEMBLE
$*      EXTENDED COMPILER DESCRIPTION FOR CMS(IML) TRANSLATION
.SYNTAX ECD
OU1 = x*S1x     .OUT(xST x x1x)  /
      x*S2x     .OUT(xST x x2x)  /
      x*S3x     .OUT(xST x x3x)  /
      x*C1x     .OUT(xCS x x1x)  /
      x*C2x     .OUT(xCS x x2x)  /
      x*C3x     .OUT(xCS x x3x)  /
      x*1x      .OUT(xGN x x1x)  /
      x*2x      .OUT(xGN x x2x)  /
      x*3x      .OUT(xGN x x3x)  /
      x*4x      .OUT(xGN x x4x)  /
      x*x       .OUT(xCI x)      /
      .STRING .OUT(xCL x *) .,
OUT = (x.OUTx x(x $ OU1 x)x  /
       x'x $ OU1 x'x  /
       x.OULBx x(x .OUT(xLB x) $ OU1 x)x  /
       x x .OUT(xLB x) $ OU1 x x  /
       x.LABELx .OUT(xLB x) OU1  /
       x.COPYLx .OUT(xCIOx)      /
       (x.ECLx / x..x) ) .OUT(xOUTx)  /
      x.OCDx x(x $ OU1 x)x  /
      x<x $ OU1 x>x  /
      x.DIAGx x(x .STRING .OUT(xMS x *) x)x  /
      (x.LBLx / x:x) .OUT(xLB x) OU1  .,
EX3 = .ID           .OUT(xCLLx *)      /
      .STRING       .OUT(xTSTx *)      /
      x.IDx         .OUT(xID x)        /
      x.LIDx x(x .STRING .OUT(xLIDx *) x)x  /
      x.COMx x(x .STRING .OUT(xCOMx *) x)x  /
      x.ERPx        .OUT(xERPx)          /
      x.HOSTx x(x .STRING .OUT(xHOSx *) x)x  /
      x.BLOFFx      .OUT(xRDBx)          /
      x.BLONx       .OUT(xNOBx)          /
      x.NUMBERx     .OUT(xNUMx)          /
      x.STRINGx     .OUT(xSR x)          /
      x(x EX1 x)x                        /
      x.EMPTYx      .OUT(xSETx)          /
      x$x .LABEL *1 EX3     .OUT(xBT x *1) .OUT(xSETx).,
EX2 = (EX3  .OUT(xBF x *1)  / OUT)
      $(EX3  .OUT(xBE x) / OUT)  .LABEL *1  .,
EX1 = EX2 $(x/x   .OUT(xBT x *1) EX2)  .LABEL *1  .,
ST  = .ID .LABEL * x=x EX1 (x.,x / x;x) .OUT(xR  x)  .,
ECD = x.SYNTAXx .ID   .OUT(xADRx *)   $ ST  x.ENDx .OUT(xENDx) .,
.END
$
```

A-6-3 COMPILER BESCHREIBUNG CMS(IML)

```
$META    *** COMPILER DESCRIPTION CMS(IML) ***
$COMPILE,INPUT
$ASSEMBLE
$*       IML SUBSET COMPILER DESCRIPTION
.SYNTAX IML
SYS   = SYSK/SYSE ;
SYSE  = xEXDCLSx ST
        'xEXT CNAF.RD,CNAF.WR,CNAF.CTR,CNAF.SJW,CNAF.SJRx' ;
SYSK  = xDCLSx ST
        'xENTRY CNAF.RD,CNAF.WR,CNAF.CTR,CNAF.SJW,CNAF.SJRx'
          xCNAF.IT0 UJP,I **x  LDAI 'xSTQ CNAF.MS1+6x'
         'xSACH CNAF.MS1+23x' CIO1 'x0 CNAF.MS1x' 'x0 8x'
         'xSEL 1001B,2x' UJM1 'xUJP CNAF.IT0x' :*S1 <xCNAF.MS1x>
         'xOCT 60545460,23214421,23603163,00602225x'
         'xOCT 31604760,13606060,60606060,60545477x'
          xCNAF.QI0 UJP,I **x  LDAI 'xSTQ CNAF.MS2+9x'
         'xSACH CNAF.MS2+35x' CIC1 'x0 CNAF.MS2x' 'x0 11x'
         'xSEL 1001B,2x' UJM1 'xUJP CNAF.QI0x' :*S1 <xCNAF.MS2x>
         'xOCT 60545460,23214421,23603163,00604624x'
         'xOCT 25516042,25314560,50602225,31604760x'
         'xOCT 13606060,60606060,60545477x'
         'xENI **,1x'  xOCT.BCD UJP **x  'xENQ 0x' 'xANA 77777Bx'
         'xSHAQ 12x' 'xSTI *-5,1x' 'xENI 3,1x' 'xSHQ 3x' 'xSHAQ 3x'
         'xIJD *-2,1x' 'xUJP CCT.BCD-1x'
          xCNAF.CTR UJP **x  OUAW EXS1 'xUJP CNAF.CTRx'
          xCNAF.SJW UJP **x  OUAW OAW EXS1 'xUJP CNAF.SJWx'
          xCNAF.SJR UJP **x  CUAW IAW EXS1 'xUJP CNAF.SJRx'
          xCNAF.WR UJP **x  OUAW OAW EXS 'xUJP CNAF.WRx'
          xCNAF.RD UJP **x  CUAW IAW EXS 'xUJP CNAF.RDx' ;
LIAI  = 'xLDA,I *-1x' 'xINA,S -1x' 'xRTJ OCT.BCDx'  ;
CIO1  = 'xEXT CIOx' 'xRTJ CIOx' 'x02 59,0x' UJM2  ;

ST    = :*S2 <x*********x> .COPYL ;
ST1   = :*S3 <x********x> ;
CONN  = 'xCON 4000B,2x' UJM1;
SEL1  = 'xSEL 10B,2x' UJM1 ;
SEL4  = 'xSEL 40B,2x' UJM1 ;
EN01  = 'xENI 0,1x' ;
EXS1  = 'xEXS 1,2x' 'xRTJ CNAF.IT0x' ;
EXS   = 'xEXS 4001B,2x' 'xRTJ CNAF.QI0x' ;
EXSQ  = 'xEXS 4000B,2x' ;
COPY  = 'xCOPY 2x' ;
NOP   = 'xNOP **x' ;
UJP2  = 'xUJP *+2x' ;
UJM5  = 'xUJP *-5x' ;
UJM2  = 'xUJP *-2x' ;
UJM1  = 'xUJP *-1x' ;
BSS1  = 'xBSS 1x' ;
OUAW  = CONN SEL1 OTAW PAUS ;
IAW   = SEL4 INAW PAUS ;
OAW   = SEL4 'xSHAQ 24x' CTAW PAUS ;
INAW  = 'xINAW 2x' UJM2;
OTAW  = 'xOTAW 2x' UJM2;
ENA   = <xENAx> ;
ANA4  = 'xANA 4000Bx' ;
STAS  = 'xSTAx *';
```

```
OCTO = 'xOCT 0x';
VFS  = <xVFD  C10/x>;
SCAS = 'xSCAx *';
SCAL = <xSCAx>;
SCAC = 'xSCAx *C1';
PAUS = 'xPAUS 4x' UJM1 ;
R    = 'xDINTx' 'xRTJ CNAF.CTRx' ;
RRD  = 'xDINTx' 'xRTJ CNAF.RDx';
RWR  = 'xDINTx' 'xRTJ CNAF.WRx';
RRDJ = 'xDINTx' 'xRTJ CNAF.SJRx';
RWRJ = 'xDINTx' 'xRTJ CNAF.SJWx';

DECL = SYS/DCLE/DCLR/DCLL ;
DCLE = xLOCDx ST .ID .LBL*
       (xHx CAAV / xPx 'xEQU *x' .ID .LBL* CAAV / xGx $ CAAV ) ;

CAAV = C N A .EOL ;
C    = NUM1 <xVFD C10/x *> / x8x VFS <x10x> / x9x VFS <x11x> ;
N    = x,x  .BLOFF <x,05/x> ( NUM4 / NUM5) .BLON ;
A    = x,x  .BLOFF <x,04/x> NUM6 .BLON ;
NUM1 = x0x/x1x/x2x/x3x/x4x/x5x/x6x/x7x ;
NUMA = x8x <x10x> / x9x <x11x> ;
NUM2 = NUMA /
       x10x <x12x> / x11x <x13x> / x12x <x14x> /
        x13x <x15x> / x14x <x16x> / x15x <x17x> ;
NUM3 = x16x <x20x> / x17x <x21x> / x18x <x22x> / x19x <x23x> /
       x20x <x24x> / x21x <x25x> / x22x <x26x> / x23x <x27x> ;
NUM4 = x24x <x30x> / x25x <x31x> / x26x <x32x> / x27x <x33x> /
       x28x <x34x> / x29x <x35x> / x30x <x36x> / x31x <x37x> ;
NUM5 = NUM3 / NUM6 ;
NUM6 = NUM2 / NUM1 <*> ;

DCLR = xLOCCx ST .ID .OCD(*S1) .ID .OCD(*S2)
       (.NUMBER .OCD(*S3) .ID .LBL* OCTO 'xDECx *C3'
       / .ID .OCD(*S3) .ID .LBL* OCTO .LBL*C3 BSS1 )
       .LBL*C2 OCTO  *C1 xEQUx *  ;
DCLL = xLOCLx ST .ID <*S1> (xSUBx :*C1 C <*S1> N A/
       xBITx x,x <xVFD Cx> BIT 'x/1x' :*C1 C <*S1> N )
       .EOL <xDECx> (.NUMBER '*' / .EMPTY '*C1') ;
BIT  = x24x<x1x>/x23x<x2x>/x22x<x3x>/x21x<x4x>/x20x<x5x>/x19x<x6x>/
       x18x<x7x>/x17x<x8x>/x16x<x9x>/x15x<x10x>/x14x<x11x>/x13x<x12x>/
       x12x<x13x>/x11x<x14x>/x10x<x15x>/x9x<x16x>/x8x<x17x>/x7x<x18x>/
       x6x<x19x>/x5x<x20x>/x4x<x21x>/x3x<x22x>/x2x<x23x>/x1x<x24x>;

IML  = .EMPTY .BLOFF PROG ;
PROG = .ERP $ STMT END REM ;
STMT = .HOST(x9   *   x)/(LID ACTS/ACTS/DECL)(x;x/M001)/.COM(x/*x) ST ;
END  = END1/END2/M002 PROG ;
END1 = xENDx x;x ST1 'xEND OF TRANSLATIONx' ;
END2 = .EOF ST1 'xEND OF FILE ENCOUNTEREDx' ;
REM  = ST1 <xVOR DEM TESTx> .REM
        'xFLASCHEN - PRESTA BITTER RED - KALTSTELLENx' ;
LID  = .LID(x:x)  * xEQU *x  ;
M001 = .DIAG(xMIC001 SEMICOLON MISSINGx) ;
M002 = .DIAG(xMIC002 NO VALID IML-KEYWORDx) ;

ACTS = (SING / MULT / BLOC / LAM / CR / SC) ;
```

```
BLOC = xUBLx ST 'xUJPx *1'  *2 xUJP **x  ENA BEE 'xUJPx *2' :*1
       .ID <*S1> ENA ( BK 'x663Bx' L0 'xRTJ CAM.ENLx' 'x40x *C1' /
            'x32Bx' .EMPTY LB 'xRTJ CAM.ENLx' 'x00x *C1')
       'x40x *2' .ID 'x00x *' 'xENA 0x' 'xSTAx *C2' 'xSTAx *C2 x+2x'
       'xEINTx' ;
BBB  = (RF SE BI RRD 'xSTAx *C1 x,1x' /
       WR SE BI 'xLDQx *C1 x,1x' RWR) REP ;
BI   = .ID <*S1> .ID <*S2> 'xLDIx * x,1x' ;
REP  = CH 'xENIx * x,1x' ;

LAM  = (xENLx  LB1 (BK 'x663Ex' L0 CAM3 / 'x32Ex' .EMPTY LB CAM1) /
        xDISLx LB1 (BK 'x667Bx' L0 CAM4 / 'x30Bx' .EMPTY LB CAM2) /
        xCLRLx LB1 (BK 'x627Ex' L0 / 'x12Ex' .EMPTY LE)) 'xEINTx' /
       xIFLx   LB1 (BK 'x701Bx' L4/'x10Bx' .EMPTY L3 LJQ) /
       xIFNLx  LB1 (BK 'x701Bx' L5/'x10Bx' .EMPTY L3 LJNQ) /
       xIFSx   LB1 (BK 'x601Ex' L4/'x33Bx' .EMPTY L3 LJQ) /
       xIFNSx  LB1 (BK 'x601Bx' L5/'x33Bx' .EMPTY L3 LJNQ) /
       (xRLSx CC0 'x601Bx' /
        xRLRx CC0 'x701Bx' /
        xRLMx CC0 'x641Bx') LB2 SIR /
       xWLMx  CC0 'x663Bx' LB2 SIW ;

LB2  = (SE/.EMPTY UJP2 :*1 C N .EOL SCAL '*1') ;
LB1  = CC0 .ID <*S1> ;
LB   = SCAC R ;
L0   = SCAC 'xLDQx *C1 x-1x' RWR ;
L1   = .ID SCAC RRD 'xEINTx' 'xLDQx *C1 x-1x' ;
L2   = .ID L3 'xANA,S 4000Bx' 'xSHAQ 11x' 'xXOA,S 1x' ;
L3   = LB 'xEINTx' ;
L4   = L1 'xAQJ,EQx *' ;
L5   = L1 'xAQJ,NEx *' ;
BK   = x( x xBITx x )x ;
CAM1 = .ID CAM2 'x00x *'/.EMPTY ;
CAM2 = 'xRTJ CAM.x *C2' 'x00x *C1' ;
CAM3 = .ID CAM4 'x00x *'/.EMPTY ;
CAM4 = 'xRTJ CAM.x *C2' 'x40x *C1' ;

CR   = (xCZx     CC0 'x34432Bx' /
        xCCx     CC0 'x34472Ex' /
        xSETCIx  CC0 'x36472Bx' /
        xCLRCIx  CC0 'x36470Ex' /
        xDISCDx  CC0 'x36530Bx' /
        xENCDx   CC0 'x36532Ex') CC2 /
       (xIFCIx   CC0 'x36473Bx' /
        xIFCDx   CC0 'x36533Ex' /
        xIFGLx   CC0 'x36573Bx') CC2 LJQ /
       (xIFNCIx CC0 'x36473Ex' /
        xIFNCDx CC0 'x36533Bx' /
        xIFNGLx CC0 'x36573Bx') CC2 LJNQ /
       xRCGLx   CC0 'x0x' CC4 SIR ;

CC0  = ST ENA ;
CC1  = (NUM1 UJP2 :*1 VFS '*'/ .EMPTY UJP2 :*1 VFS NUMA ..) SCAL '*1' ;
CC2  = CC1 R 'xEINTx' ;
CC3  = CC2 COPY ;
CC4  =( SE / .EMPTY UJP2 :*1 C <x,05/36,04/2x> .EOL SCAL '*1') ;
IF1  = ANA4 'xAZJ,EQx *' ;
IF0  = ANA4 'xAZJ,NEx *' ;
```

```
SC    = xEZx     CC0 'x432Bx' R 'xEINTx' /
        xRBCRx   CC0 'x0x'    SCE SIR /
        xWBCRx   CC0 'x20Bx' SCE SIW /
        xRBQx    CD1 ANA4 STAS /
        xIFBQx   CD1 IF1 /
        xIFNBQx  CD1 IF0 /
        (xENBDx   CD3 'xSEL 0,2x' /
         xDISBDx CD3 'xSEL 1,2x' ) UJM1 'xEINTx' ;

SCE   = (SE / .EMPTY UJP2 :*1 <xVFD 019/x> NUM6 .ECL SCAL '*1') ;
CD1   = CD3 .ID 'xEINTx' CCPY ;
CD3   = ST 'xDINTx' CONN ;

SING = xSJNQx CC0 SSJ LJNQ /
       xSJQx  CC0 SSJ LJQ /
       xSAx   CC0 SSS ;

SSJ   = (WR SE .ID 'xLDQx *' RWRJ /
         OP SE R /
         RF SE .ID RRDJ STAS) 'xEINTx' ;
SSS   = WR SE SIW /
        OP SE R 'xEINTx' /
        RF SE SIR ;
SE    = .ID SCAS ;
SIR   = .ID RRD 'xEINTx' STAS ;
SIW   = .ID 'xLDQx *' RWR 'xEINTx' ;
LJNQ  = .ID EXSQ 'xUJPx *' ;
LJQ   = .ID EXSQ UJP2 'xUJPx *' ;

MULT = xMAx ST EN01 .LBL*1 ENA MMM .ID CH 'xLDAQx *' 'xAQJ,NEx *1' ;
MMM   = WR ME .ID 'xLDQx * x,1x' RWR /
        OP ME R /
        RF ME .ID RRD 'xSTAx * x,1x' ;
ME    = .ID 'xSCAx * x,1x' ;
CH    = 'xEINTx' COPY 'xSTAx * x+2x' 'xINI 1,1x' 'xSTIx * x,1x' ;

RF    = ((xREAD2x/xF1x) <x1x> / (xREADx/xF0x) <x0x> /
         (xRCOMPx/xF3x) <x2x> / (xRCLx /xF4x) <x4x> /
         xF5x <x5x> / xF6x <x6x> / xF7x <x7x> ) 'xBx' ;
WR    = ((xWRIT2x/xF17x) <x21x> / (xWRITEx/xF16x) <x20x> /
         (xBSET2x/xF19x) <x23x> / (xBISETx/xF22x) <x22x> /
         (xBCLR2x/xF23x) <x27x> / (xBICLRx/xF21x) <x25x> /
         xF20x <x24x> / xF22x <x26x> ) 'xBx' ;
OP    = ((xTLAMx  /xF8x ) <x10x> / (xTSTATx/xF27x) <x33x> /
         (xCLR2x/xF11x) <x13x> / (xCLEARx/xF9x ) <x11x> /
         xF10x <x12x> / xF12x <x14x> / xF13x <x15x> /
         xF14x <x16x> / xF15x <x17x> /
         (xENABLx/xF26x) <x32x> / (xDISABx/xF24x) <x30x> /
         (xEXx    /xF25x) <x31x> / xF28x <x34x> /
         xF29x <x35x> / xF30x <x36x> / xF31x <x37x> ) 'xBx' ;
.END
$
```

A-6-4 BACKUP - BEISPIEL

```
$META,MLOAD    *** LOADED VERSION EX9 ***
$COMPILE,INPUT
$ASSEMBLE
$*      EXTENDED COMPILER DESCRIPTION FOR IML TRANSLATION
.SYNTAX ECD
OU1 = x*S1x     .OUT(xST x x1x)  /
      x*S2x     .OUT(xST x x2x)  /
      x*S3x     .OUT(xST x x3x)  /
      x*C1x     .OUT(xCS x x1x)  /
      x*C2x     .OUT(xCS x x2x)  /
      x*C3x     .OUT(xCS x x3x)  /
      x*1x      .OUT(xGN x x1x)  /
      x*2x      .OUT(xGN x x2x)  /
      x*3x      .OUT(xGN x x3x)  /
      x*4x      .OUT(xGN x x4x)  /
      x*x       .OUT(xCI x)      /
      .STRING .OUT(xCL x *) .,
OUT = (x.OUTx x(x $ OU1 x)x  /
       x'x $ OU1 x'x  /
       x.OULBx x(x .OUT(xLB x) $ OU1 x)x  /
       x x .OUT(xLB x) $ OU1 x x  /
       x.LABELx .OUT(xLB x) OU1  /
       (x.EOLx / x..x) ) .OUT(xOUTx)  /
      x.OCDx x(x $ OU1 x)x  /
      x<x $ OU1 x>x  /
      (x.LBLx / x:x) .OUT(xLB x) OU1  .,
EX3 = .ID           .OUT(xCLLx *)      /
      .STRING       .OUT(xTSTx *)      /
      x.IDx         .OUT(xID x)          /
      x.LIDx        .OUT(xLIDx x : x)    /
      x.COMx        .OUT(xCOMx x /* x)   /
      x.ERPx        .OUT(xERPx)          /
      x.BLOFFx      .OUT(xRDBx)          /
      x.BLONx       .OUT(xNOBx)          /
      x.NUMBERx     .OUT(xNUMx)          /
      x.STRINGx     .OUT(xSR x)          /
      x(x EX1 x)x                        /
      x.BACKUPx x(x.OUT(xUP x) EX1 x)x /
      x.EMPTYx       .OUT(xSETx)         /
      x$x .LABEL *1 EX3     .OUT(xBT x *1) .OUT(xSETx).,
EX2 = (EX3  .OUT(xBF x *1)  / OUT)
      $ (EX3 .OUT(xBE x) .OUT(xBF x *1) / OUT) .LABEL *1  .,
EX1 = EX2 $(x/x    .OUT(xBT x *1) EX2)  .LABEL *1  .,
ST  = .ID .LABEL * x=x EX1 (x.,x / x;x) .OUT(xR  x)  .,
ECD = x.SYNTAXx .ID    .OUT(xADRx *)   $ ST  x.ENDx .OUT(xENDx) .,
.END
$
$META    *** TEST FOR BACKUP ***
$COMPILE,INPUT,TRACE
$ASSEMBLE,LIST
.SYNTAX TEST
TEST = x.STARTx '*' $($ ST1 x;x) x.ENDx '*' ;
ST1  = .BACKUP(.ID x:x .OUT(xLABELx)) / .ID .OUT(xIDENTIFIERx) ;
.END
$
```

```
$META    *** TEST EXAMPLE FOR BACKUP  ***
$COMPILE,INPUT,TRACE,FTRACE,LIST
.START
IDENTIFIER ;
LABEL: ;
LABEL: IDENTIFIER ;
.END
$
$$
```

A-7 META-II/X Listing

```
C     ----------------------
C     -   M E T A  I I / X  -
C     ----------------------
C     EXAMPLE OF A SYNTAX ORIENTED METACOMPILER BASED ON META II
C     PRINCIPLES BY D.V. SCHORRE (PROC ACM 19TH NATL. CONF
C     1964, PAGE D1.3)
C     WRITTEN BY WILFRIED KNEIS, GFK, KARLSRUHE, ZYKLOTRONLABOR
C     THIS VERSION OF META II HAS THE SAME BASIC CAPABILITIES AS THE
C     ORIGINAL SCHORRE VERSION HOWEVER WITH ADDITIONAL EXTENSIONS
C
C     THE EXTENSIONS ARE
C     ------------------
C     I.  MAINPROGRAM
C         PROGRAM AND INPUT/OUTPUT CONTROL BY '$.....'-CONTROL CARDS
C         A. META CARD
C            TO LOAD THE SUITED COMPILER VERSION FROM PERIPHEREL OR
C            IF NOT SPECIFIED FROM COMMON AREA INTO THE SIMULATOR
C            PARAMETERS:  MLOAD - TO LOAD COMPILER IN MACHINE CODE FORM
C                         NLOAD - TO LOAD COMPILER IN MNEMONIC ASSEMBLER
C         B. COMPILE CARD
C            TO SPECIFY THE COMPILE STEP (SIMULATOR RUN)
C            PARAMETERS:  INPUT - LIST OF INPUT TO SIMULATOR
C                         TRACE - PROMT SIMULATOR OUTPUT LISTING
C                         LIST  - SIMULATOR OUTPUT LISTING  IN
C                                 CDC-COMPASS-ASSEMBLER FORMAT
C                                 THIS OUTPUT ONLY WILL OCCUR, WHEN
C                                 NO ERROR HAS OCCURED IN THE COMPILE
C                                 STEP
C                         FTRACE- COMPLETE SIMULATOR TRACE
C                         CARD  - OUTPUT IN COMPASS CARD IMAGES
C         C. ASSEMBLE CARD
C            TO SPECIFY THE ASSEMBLE STEP
C            AND TO SAVE THE SUITED COMPILER VERSION, IF SPECIFIED
C            PARAMETERS:  LIST  - ASSEMBLY OUTPUT LISTING ON PRINTER
C                         MSAVE - TO SAVE COMPILER IN MACHINE CODE FORM
C                         NSAVE - TO SAVE COMPILER IN MNEMONIC ASSEMBLER
C         D. COMMENT CARD
C            PARAMETERS:  NONE
C         E. RUN CARD
C            TO BEGIN EXECUTION CYCLE AND RUN WITH SPECIFIED OPTIONS
C            PARAMETERS:  NONE
C         F. END CARD
C            TO END THE WHOLE RUN - STOP
C             PARAMETERS:  NONE
C
C         EXAMPLES OF CONTROL CARDS
C     META CARD:
C         $META,MLOAD,NLOAD
C     COMPILE CARD:
C         $COMPILE,INPUT,TRACE(A,B),LIST(A,B),FULLTRACE(A,B),CARD
C     ASSEMBLE CARD:
C         $ASSEMBLE,LIST(A,B),MSAVE,NSAVE
C     COMMENT CARD
C         $*  ........
C     RUN CARD:
C         $
```

```
C      END CARD:
C          $$
C
C      II. SIMULATOR
C
C          A. GENERAL DIAGNOSTIC AND BACKUP FUNCTIONS (BE), (ERP), (UP)
C             1. STOP MODE
C             2. RETURN TO ERP AND PROCESS NEXT STATEMENT
C             3. BACKUP IF UP IS ON
C          B. LABEL BOX FUNCTIONS (GN)
C             FOUR INDEPENDENT STACK LABEL BOXES AVAILABLE AT EVERY STAGE
C             IN RECURSIVE CALLING
C          C. UNIVERSAL REGISTER FUNCTIONS (ST), (CS), (CLS)
C             THREE COMMON NONSTACK REGISTERS WITH VARIABLE LENGTHS (MAX.
C             73 WORDS), AVAILABLE FROM EVERY STACK DEEP.
C          D. COMMENT FUNCTION (COM)
C             CHECKS COMMENTS OF THE FORM /* COMMENT /*
C          E. SPECIAL LABEL FUNCTION (LID)
C             CHECKS LABELS OF THE FORM .ID  :  IN THE NON-BACKUP MODE
C          F. DIAGNOSTIC MESSAGE FUNCTION (MS)
C             TO OUTPUT SPECIAL MESSAGES DURING PARSING
C          G. HOST LANGUAGE FUNCTION (HOS)
C             FOR PARSING IN AN ASSEMBLER (COMPASS ASSEMBLER) LANGUAGE
C             ENVIRONMENT
C          H. INPUT LINE LIST FUNCTION (CIO)
C             FOR LISTING INPUT LINE AS COMMENT LINE IN OBJECT FILE
C             (ON FILES ISOUT, IPRI, ICCD)
C
C      III. I/O - CONVENTIONS
C          ICCD =        = FT02F001 - COMPASS CARD OUTPUT
C          IFILE=        = FT04F001 - INPUT FILE
C          ISIN = SYSIN  = FT05F001 - STANDARD INPUT
C          ISOUT= SYSOUT = FT06F001 - PROMPT STANDARD OUTPUT
C          IPRI =        = FT07F001 - STANDARD OUTPUT FOR SIMULATOR
C          IPUN =        = FT08F001 - DUMPOUTPUT
C          ICAR =        = FT09F001 - DUMPINPUT
C
C      IV. MISCELLENIOUS TOPICS
C          1. UNIFORM MESSAGE FORMAT
C          2. ERROR DIAGNOSTIC FOR SYSTEM LIMITS PROVIDED
C          3. ERROR DIAGNOSTICS IN SUBROUTINE ASEMBL
C          4. MACHINE INDEPENDENT STRING REPRESENTATION
C             OF THE ORIGINAL VERSION IN BLOCK DATA
C          5. MACHINE INDEPENDENT ASSEMBLER OUTPUT ON
C             FILE IPUN
C          6. MACHINE DEPENDENT MACHINE OUTPUT ON FILE IPUN
C
       IMPLICIT INTEGER(A-Z)
       COMMON /LIMITS/ CIL,COL,COL1,COL2,CDL,CDL1,CDL2,ICYCLS,
      1                CTL,CTL1,CTL2,ALL,ALL1,ALL2,MSAVE,NSAVE,NCARD,
      2                IND,JCNTR,BRA,COMMA,KET,CHMASK,
      3                NA,NI,NJ,NR,NS,NULL,NINE,
      4                KOPS,IADR,NMOP(40),IEND,IEL,IEQUAL,ISTERN,IPER,
      5                ISTR,LOPF,LADRF
       COMMON /INPSIM/ LEFF,LMAX,LONE,LTEN,LHUN,LALPHA,LASTP,LOP,LOPND,
      1                ISYMB(6),ICSYM(4),IASYM(4),IA00
       COMMON /ORIGIN/ MAXOUT,MCLIM,LENGTH,LMASK,KODE(4000)
       COMMON /ASSEM/ DDATE,DTIME,IASS(9500),ICNT
```

```
      COMMON /META/ IP,MCODE(4000),LBL(4),INBUF(80),IROW,S,MROW,OUT(80),
     1  OROW,ICCD,IFILE,ISIN,ISOUT,IPRI,IPUN,ICAR,IPAGE,ICOUNT,LNBR,UP
     2  ,INLN,CUTUP,ERR,EFLG,ECNT
      LOGICAL JCL,LOAD,SAVE,COMP,ASS,CIL,CDL,CCL,CTL,ALL,MSAVE,NSAVE
      LOGICAL MLOAD,NLOAD,ICOMP,ASSF,IMLF,NCARD,EFLG
      LOGICAL*1 LCARD(1),LINBUF(1)
      REAL TOTAL,TOTALO,RTIME,RTIMEO,CTIME,CTIMEO,ATIME,ATIMEO
      REAL TAV,RAV,CAV,AAV,ZEIT
      REAL*8 DDATE,DTIME
      EQUIVALENCE (INBUF(1),LCARD(1)),(INBUF(1),LINBUF(1))
      DATA TOTAL,TOTALO,RTIME,RTIMEO,CTIME,CTIMEO,ATIME,ATIMEO
     1     /8*0./
C     INITIAL AND RE-INITIAL VALUES
C
      CALL IATUM(DDATE,DTIME)
      IPAGE=1
      JCL=.TRUE.
      COMP=.FALSE.
      ASS=.FALSE.
      LOAD=.FALSE.
      SAVE=.FALSE.
      MCLIM=(4000)
      TOTALO=ZEIT(0)
 1000 KMAX=LMAX
      LMAX=0
      MAXOUT=(9500)-20
      ICOUNT=0
      ILINE=0
      CIL=.FALSE.
      COL=.FALSE.
      CDL=.FALSE.
      CTL=.FALSE.
      ALL=.FALSE.
      MLOAD=.FALSE.
      NCARD=.FALSE.
      NLOAD=.FALSE.
      MSAVE=.FALSE.
      NSAVE=.FALSE.
      LONE=0
      LTEN=0
      LHUN=0
      LALPHA=IA00
      GOTO 1300
C
C     CENTRAL INPUT AND INPUTLISTING - PROGRAM CARD LOOP
C
 1100 IF(.NOT.(JCL .OR. CIL)) GOTO 1300
      IF(ICOUNT .GT. 1) GOTO 1220
      WRITE(ISOUT,1210) DDATE,IPAGE
 1210 FORMAT(1H1,/,' META II/X - COMPILER -INPUT',T50,'DATE ',A8,
     1       T78,'PAGE ',I3,/)
      IPAGE=IPAGE+1
 1220 IF(ICOUNT .EQ. 55 ) ICOUNT=0
      WRITE(ISOUT,1230) INBUF
 1230 FORMAT(5X,80A1)
 1300 ICOUNT=ICOUNT+1
      READ(ISIN,1310,END=580) (INBUF(I),I=1,80)
 1310 FORMAT(80A1)
```

```
      IF(INBUF(1) .EQ. JCNTR) GOTO 1600
 1400 JCL=.FALSE.
      LMAX=LMAX+1
      WRITE(IFILE,1310) (INBUF(I),I=1,80)
      IF(LMAX .LE. 4200) GOTO 1100
      WRITE(ISOUT,1430)
 1430 FORMAT(/,' M100 *** INPUT FILE, 4200 CARDS, TOC SMALL ***')
      STOP 100
C
C     CONTROL CARD LOOP
C
 1600 JCL=.TRUE.
      IND=2
      ITYP=0
      DO 1610 I=1,6
      IF(INBUF(2) .EQ. ISYMB(I)) ITYP=I
 1610 CONTINUE
      IND=NEXT(COMMA)
 1620 IF(IND .GT. 72) GOTO 1100
      ICH=INBUF(IND+1)
 1630 GOTO(100,200,300,1100,500,600),ITYP
      STOP 130
C
C     META CARD PROCESSOR
C
  100 JCL=.TRUE.
      LOAD=.TRUE.
      DO 104 I=1,4
      IF(ICH .EQ. IASYM(I)) GOTO (110,120,130,110),I
  104 CONTINUE
  110 IND=IND+1
      GOTO 1620
  120 CALL CCPARM(MLCAD,IDY1,IDY2)
      GOTO 1620
  130 CALL CCPARM(NLCAD,IDY1,IDY2)
      GOTO 1620
C
C     COMPILE CARD PROCESSOR
C
  200 COMP=.TRUE.
      DO 204 I=1,4
      IF(ICH .EQ. ICSYM(I)) GOTO (210,220,230,240),I
  204 CONTINUE
      IF(ICH .EQ. IASYM(4)) GOTO 250
      IND=IND+1
      GOTO 1620
  210 CALL CCPARM(CIL,IDY1,IDY2)
      GOTO 1620
  220 CALL CCPARM(COL,COL1,COL2)
      GOTO 1620
  230 CALL CCPARM(CDL,CDL1,CDL2)
      GOTO 1620
  240 CALL CCPARM(CTL,CTL1,CTL2)
      GOTO 1620
  250 CALL CCPARM(NCARD,IDY1,IDY2)
      GOTO 1620
C
C     ASSEMBLE CARD PROCESSOR
```

```
C
  300 ASS=.TRUE.
      DO 304 I=1,4
      IF(ICH .EQ. IASYM(I)) GOTO(310,320,330,304),I
  304 CONTINUE
      IND=IND+1
      GOTO 1620
  310 CALL CCPARM(ALL,ALL1,ALL2)
      GOTO 1620
  320 CALL CCPARM(MSAVE,IDY1,IDY2)
      GOTO 350
  330 CALL CCPARM(NSAVE,IDY1,IDY2)
  350 SAVE=.TRUE.
      GOTO 1620
C
C     END CARD PROCESSOR (ACCOUNTING)
C
  500 CONTINUE
      WRITE(ISOUT,1230) INBUF
      WRITE(ISOUT,510)
  510 FORMAT(/,' M140 *** END OF INPUT ***')
      TOTAL=ZEIT(TOTAL0)
      RTIME=TOTAL-CTIME-ATIME
      TAV=TOTAL*1000/KMAX
      CAV=CTIME*1000/KMAX
      AAV=ATIME*1000/KMAX
      RAV=RTIME*1000/KMAX
      WRITE(ISOUT,520) KMAX,TOTAL,RTIME,CTIME,ATIME,TAV,RAV,CAV,AAV
  520 FORMAT(1H1,/,' ACCOUNTING',/,' **********',/,
     / ' NUMBER OF CARDS',T35,I5,/,
     1 ' TOTAL TIME',T35,F5.2,' SEC',/,
     2 ' FILE HANDLING TIME',T35,F5.2,' SEC',/,
     3 ' COMPILE TIME',T35,F5.2,' SEC',/,
     4 ' ASSEMBLE TIME',T35,F5.2,' SEC',//,
     5 ' AVERAGE TOTAL TIME',T35,F4.0,' MSEC',/,
     6 ' AVERAGE FILE HANDLING TIME',T35,F4.0,' MSEC',/,
     7 ' AVERAGE COMPILE TIME',T35,F4.0,' MSEC',/,
     8 ' AVERAGE ASSEMBLE TIME',T35,F4.0,' MSEC')
  580 WRITE(ISOUT,590)
  590 FORMAT(/,' M150 *** END OF JOB ***',/,1H1)
      STOP 150
C
C     RUNCARD PROCESSOR - EXECUTION CYCLE
C
  600 WRITE(ISOUT,1230) INBUF
  605 IETYP=5
      IF(SAVE)   IETYP=4
      IF(ASS)    IETYP=3
      IF(COMP)   IETYP=2
      IF(LOAD)   IETYP=1
      GOTO(610,620,630,640,1000),IETYP
C
  610 IF(.NOT. NLOAD) GOTO 614
      READ(ICAR,1310) INBUF
      READ(ICAR,648) ICNT
      READ(ICAR,643) (IASS(I),I=1,ICNT)
      WRITE(ISOUT,612) (INBUF(I),I=1,72)
  612 FORMAT(/,' M161 ***',72A1,'HAS BEEN LOADED INTO IASS ***')
```

```
      IF(ICNT .LT. MAXOUT) GOTO 618
      WRITE(ISOUT,611)
  611 FORMAT(/,' M160 *** MACHINE BUFFER, IASS(9500), TOO SMALL ***')
      STOP 160
  614 IF(.NOT. MLOAD) GOTO 616
      READ(ICAR,1310) INBUF
      READ(ICAR,642) LENGTH,(KODE(I),I=1,LENGTH)
      WRITE(ISOUT,613) (INBUF(I),I=1,72)
  613 FORMAT(/,' M171 ***',72A1,'HAS BEEN LOADED INTO KODE ***')
      IF(LENGTH .LT. MCLIM) GOTO 616
      WRITE(ISOUT,615)
  615 FORMAT(/,' M170 *** MACHINE BUFFER, MCODE(4000), TOO SMALL ***')
      STOP 170
  616 DO 617 I=1,LENGTH
  617 MCODE(I)=KODE(I)
  618 LOAD=.FALSE.
      GOTO 605
  620 ENDFILE IFILE
      REWIND IFILE
      CTIME0=ZEIT(0)
      CALL SIM
      CTIME=ZEIT(CTIME0)
      REWIND IFILE
      WRITE(ISOUT,622) ECNT,ICNT
  622 FORMAT(/,' M180 *** COMPILE STEP: ',I6,' DIAGNOSTICS, PROGRAMLENGT
     1H =',I6,' ***')
      COMP=.FALSE.
      IF(EFLG) STOP 181
      GOTO 605
  630 ATIME0=ZEIT(0)
      CALL ASEMBL
      ATIME=ZEIT(ATIME0)
      WRITE(ISOUT,632) ECNT,LENGTH
  632 FORMAT(/,' M190 *** ASSEMBLE STEP:',I6,' DIAGNOSTICS, PROGRAMLENGT
     1H =',I6,' ***')
      ASS=.FALSE.
      IF(EFLG) STOP 191
      GOTO 605
  640 WRITE(IPUN,641)
  641 FORMAT(80X)
      IF(.NOT. MSAVE) GOTO 644
      WRITE(IPUN,642) LENGTH,(KODE(I),I=1,LENGTH)
      WRITE(ISOUT,646)
  646 FORMAT(/,' M200 *** MSAVE EXECUTED, KODE SAVED ON IPUN ***')
  642 FORMAT(9I8,8X)
  644 IF(.NOT. NSAVE) GOTO 645
      WRITE(IPUN,648) ICNT
  648 FORMAT(I12)
      WRITE(IPUN,643) (IASS(I),I=1,ICNT)
      WRITE(ISOUT,647)
  647 FORMAT(/,' M210 *** NSAVE EXECUTED, IASS SAVED ON IPUN ***')
  643 FORMAT(18A4,8X)
  645 END FILE IPUN
      REWIND IPUN
      SAVE=.FALSE.
      GOTO 605
      END
```

```
      SUBROUTINE CCPARM( LOG,LOG1,LOG2)
      IMPLICIT INTEGER (A-Z)
      COMMON /LIMITS/ CIL,COL,COL1,COL2,CDL,CDL1,CDL2,ICYCLS,
     1                CTL,CTL1,CTL2,ALL,ALL1,ALL2,MSAVE,NSAVE,NCARD,
     2                IND,JCNTR,BRA,COMMA,KET,CHMASK,
     3                NA,NI,NJ,NR,NS,NZ,NULL,NINE,
     4                KOPS,IADR,NMOP(40),IEND,IBL,IEQUAL,ISTERN,IPER,
     5                ISTR,LCPF,LADRF
      COMMON /ORIGIN/ MAXOUT,MCLIM,LENGTH,LMASK,KODE(4000)
      CCMMON /META/ IP,MCODE(4000),LBL(4),INBUF(80),IROW,S,MROW,OUT(80),
     1  OROW,ICCD,IFILE,ISIN,ISOUT,IPRI,IPUN,ICAR,IPAGE,ICOUNT,LNBR,UP
     2  ,INLN,OUTUP,ERR,EFLG,ECNT
      LOGICAL LOG,LETTER,DIGIT
      LETTER(I)=      I .GE. NA .AND. I .LE. NI
     1           .OR. I .GE. NJ .AND. I .LE. NR
     2           .OR. I .GE. NS .AND. I .LE. NZ
      DIGIT(I)=I .GE. NULL .AND. I .LE. NINE
C
C     DEMAND OF COMMA - NEXT SYMBOL CAN BE A LETTER, IF YES, EXIT
C
      LOG=.TRUE.
      LOG1=0
      LOG2=1000
   10 IND=NEXT(CCMMA)
      I=IND+1
      IF(LETTER(INBUF(I)) .OR. IND .GE. 72) RETURN
C
C     FIRST NUMBER CONVERSION UNTIL BRA RECOGNIZED
   20 KIND=IND
      I=0
   30 IND=IND-1
      IF(INBUF(IND) .EQ. BRA) GOTO 40
      IF(.NOT. DIGIT(INBUF(IND))) STOP 310
      LOG1=LOG1+((INBUF(IND)-CHMASK)/LMASK**3)*10**I
      I=I+1
      GOTO 30
C
C     SECOND NUMBER CONVERSION UNTIL COMMA RECOGNIZED
   40 IND=KIND+1
   42 IF(INBUF(IND) .EQ. KET) GOTO 50
      IND=IND+1
      GOTO 42
   50 KIND=IND
      NUM=0
      I=0
   60 IND=IND-1
      IF(INBUF(IND) .EQ. COMMA) GOTO 70
      IF(.NOT. DIGIT(INBUF(IND))) STOP 320
      NUM=NUM+((INBUF(IND)-CHMASK)/LMASK**3)*10**I
      I=I+1
      GOTO 60
   70 IND=KIND
      IF(NUM .NE. 0) LOG2=NUM
      GOTO 10
      END
```

W. Kneis, META-II/X-System

```
      INTEGER FUNCTION NEXT(SYMBOL)
      IMPLICIT INTEGER (A-Z)
      COMMON /LIMITS/ CIL,COL,COL1,COL2,CDL,CDL1,CDL2,ICYCLS,
     1                CTL,CTL1,CTL2,ALL,ALL1,ALL2,MSAVE,NSAVE,NCARD,
     2                IND
      COMMON /META/ IP,MCODE(4000),LBL(4),INBUF(80)
      NEXT=IND+1
   10 IF(INBUF(NEXT) .EQ. SYMBOL .OR. NEXT .GE. 72) RETURN
      NEXT=NEXT+1
      GOTO 10
      END
```

```
      BLOCK DATA
C     KODORG = STARTING VERSION OF THE META II/X COMPILER REPRESENTATION
C     LENGTH = LENGTH OF THE META II/X  COMPILER REPRESENTATION
      IMPLICIT INTEGER (A-Z)
      COMMON /ORIGIN/ MAXOUT,MCLIM,LENGTH,LMASK,KODORG(4000)
      COMMON /LIMITS/ CIL,COL,COL1,COL2,CDL,CDL1,CDL2,ICYCLS,
     1                CTL,CTL1,CTL2,ALL,ALL1,ALL2,MSAVE,NSAVE,NCARD,
     2                IND,JCNTR,BRA,COMMA,KET,CHMASK,
     3                NA,NI,NJ,NR,NS,NZ,NULL,NINE,
     4                KOPS,IADR,NMOP(40),IEND,IBL,IEQUAL,ISTERN,IPER,
     5               ISTR,LOPF,LADRF
      COMMON /INPSIM/ LEFF,LMAX,LONE,LTEN,LHUN,LALPHA,LASTP,LOP,LOPND,
     1                ISYMB(6),ICSYM(4),IASYM(4),IA00
      COMMON /META/ IP,MCODE(4000),LBL(4),INBUF(80),IROW,S,MROW,OUT(80),
     1  OROW,ICCD,IFILE,ISIN,ISOUT,IPRI,IPUN,ICAR,IPAGE,ICOUNT,LNBR,UP
     2  ,INLN,CUTUP,ERR,EFLG,ECNT
      DATA JCNTR,BRA,COMMA,KET,CHMASK/'$','(',',',')',ZF0404040/
      DATA ISYMB /'M','C','A','*','$',' '/
      DATA ICSYM /'I','T','L','F'/
      DATA IASYM /'L','M','N','C'/
      DATA MCLIM /2000/,IA00 /'A000'/,LMASK /256/
      DATA ICCD,IFILE,ISIN,ISOUT,IPRI,IPUN,ICAR /2,4,5,6,7,8,9/
C     PROGRAM LENGTH =LENGTH (FROM MESSAGE NO. M190)
      DATA LENGTH /188/
C     .SYNTAX PRO
      DATA KODORG /167,
C     OU1 = ...
     /204,'*1',1012,315,'GN',115,'1',14,2410,104,'*',1712,315,'CI',14,
     /2410,7,2412,315,'CL',16,14,9,
C     OUT = ...
     /404,'.OUT',3812,104,'(',13,208,3110,1,13,104,')',13,4810,604,
     /'.LAB','EL',4812,315,'LB',14,208,13,5212,315,'OUT',14,9,
C     EX3 = ...
     /5,5912,315,'CLL',16,14,10610,7,6612,315,'TST',16,14,10610,304,
     /'.ID',7312,315,'ID',14,10610,704,'.STR','ING',8112,315,'SR',
     /14,10610,104,'(',9012,13208,13,104,')',13,10610,104,'$',10612,
     /21,117,14,5308,13,315,'BT',117,14,315,'SET',14,9,
C     EX2 = ...
     /5308,11312,315,'BF',117,14,11610,2508,11612,13112,5308,12212,
     /315,'BE',14,12510,2508,12512,11710,1,13,21,117,14,9,
C     EX1 = ...
     /10708,14912,104,'/',14312,315,'BT',117,14,10708,13,13410,1,
     /13,21,117,14,9,
C     ST  = ...
     /5,16612,21,16,14,104,'=',13,13208,13,204,'.,',13,315,'R',14,9,
C     PRO = ...
     /704,'.SYN','TAX',18712,5,13,315,'ADR',16,14,15008,17710,1,
     /13,404,'.END',13,315,'END',14,9 /
      DATA NA,NI,NJ,NR,NS,NZ,NULL,NINE /'A','I','J','R','S','Z','0','9'/
      DATA KOPS,IADR,IEND,IBL,IEQUAL,ISTERN /40,'ADR','END',' ',' ','*'/
      DATA IPER,ISTR,LOPF,LADRF/'.','-',10,20/
      DATA NMOP /'SET','NOB','RDB','TST','ID','NUM','SR',
     1   'CLL','R','BT','B','BF','BE','OUT','CL','CI','GN',
     2   'COM','LIT','ERR','LB','MS','ST','CLS','HOS',
     3   'CS','UP','CIT','EOF','FOP','OP','CIP','CLP','POT','HV','HLV',
     4   'REM','XYZ','XYZ','XYZ'/
      DATA LASTP,LOP,LOPND/0,40,0/
      END
```

W. Kneis, META-II/X-System

```
      SUBROUTINE SIM
C     MEANING OF THE VARIABLES
C     INBUF = INPUT BUFFER
C     IROW = POINTER OF INPUT BUFFER, INBUF
C     MROW = MARKER OF IROW
C     MCODE = INTERNAL COMPILER REPRESENTATION
C     MCLIM = DELIMITER OF COMPILER BUFFER, MCODE
C     MPOINT = LABEL STACK POINTER
C     OUT = OUTPUT BUFFER
C     OROW = MARKER OF OUTPUT BUFFER FOR LABELS AND OP CODES
C     P = PROGRAM COUNTER
C     OPND = OPERAND
C     S = MAIN SWITCH
C     M = BLANK MODE SWITCH
C     UP = BACKUP SWITCH
C     ERR = ERROR SWITCH
C     EFLG = ERROR FLAG
C     ECNT = ERROR COUNT
C     ICYCLS = LINE NUMBER FOR FULLTRACE ON
C     LNBR = LINE NUMBER FOR TRACE ON
      IMPLICIT INTEGER (A-Z)
      REAL*8 DDATE,DTIME
      COMMON /INPSIM/ LEFF,LMAX,LONE,LTEN,LHUN,LALPHA,LASTP,LOP,LOPND,
     1                ISYMB(6),ICSYM(4),IASYM(4),IA00
      COMMON /ORIGIN/ MAXOUT,MCLIM,LENGTH,LMASK,KODE(4000)
      COMMON /ASSEM/ DDATE,DTIME,IASS(9500),ICNT
      COMMON /LIMITS/ CIL,CCL,COL1,COL2,CDL,CDL1,CDL2,ICYCLS,
     1                CTL,CTL1,CTL2,ALL,ALL1,ALL2,MSAVE,NSAVE,NCARD,
     2                IND,JCNTR,BRA,COMMA,KET,CHMASK,
     3                NA,NI,NJ,NR,NS,NZ,NULL,NINE,
     4                KOPS,IADR,NMOP(40),IEND,IBL,IEQUAL,ISTERN,IPER,
     5                ISTR,LOPF,LADRF
      COMMON /META/  P,MCODE(4000),LBL(4),INBUF(80),IROW,S,MROW,OUT(80),
     1  OROW,ICCD,IFILE,ISIN,ISOUT,IPRI,IPUN,ICAR,IPAGE,ICOUNT,LNBR,UP
     2  ,INLN,OUTUP,ERR,EFLG,ECNT
      DIMENSION IPOINT(80),STREG(3,80),MARSK(3),MS(80)
      LOGICAL CIL,CDL,COL,CTL,ALL,MSAVE,NSAVE,NCARD
      LOGICAL S,M,LETTER,DIGIT,UP,ERR,EFLG,LIPRI,LTRACE,LISOUT
      LOGICAL*1 LMCODE(1),LLITL,LLBL(1),LAS(1),LOUT(1),LIREM(1)
      EQUIVALENCE (MCODE(1),LMCODE(1)),(LITL,LLITL),(IASS(1),LAS(1))
     1           ,(LBL(1),LLBL(1)),(OUT(1),LOUT(1)),(IREM,LIREM(1))
      DATA REM /' 000'/
C
      LETTER(I) = I .GE. NA .AND. I .LE. NI
     1       .OR. I .GE. NJ .AND. I .LE. NR
     2       .OR. I .GE. NS .AND. I .LE. NZ
      DIGIT(I)  = I .GE. NULL .AND. I .LE. NINE
      LISOUT(LNBR) = COL .AND. (LNBR .GE. COL1 .AND. LNBR .LE. COL2)
      LIPRI(LNBR)  = (CDL .AND. (LNBR .GE. CDL1 .AND. LNBR .LE. CDL2))
     1                .OR. NCARD
      LTRACE(ICYCLS) = CTL .AND. ICYCLS .GE. CTL1 .AND. ICYCLS .LE. CTL2
C
C     INITIAL VALUES
C
      LNBR=0
      LEFF=0
      LITL=IBL
      S = .FALSE.
```

```
      M = .TRUE.
      UP=.FALSE.
      ERR=.FALSE.
      EFLG=.FALSE.
      ECNT=0
      ICYCLS=0
      IROW=73
      MROW=72
      ICNT=1
      MPOINT=MCLIM
      OROW=7
      DO 2 I=1,80
      INBUF(I)=IBL
      IPOINT(I)=IBL
    2 OUT(I)=IBL
      IF(COL .OR. CTL) WRITE(ISCUT,3) DDATE
    3 FORMAT(1H1,/,' META II/X - SIMULATOR - TRACE - OUTPUT',T50,
     1'DATE ',A8,//,
     2' ICYCLS  P',4X,'OP NMOP',2X,'OPND',3X,'S',4X,'M',2X,
     3'MPOINT',1X,'IROW',3X,'MCODE(MPOINT-MPOINT+5)',6X,
     4'INBUF(MROW-IROW)')
C
C     INTERPRETIVE CYCLE - TRACE OUTPUT
C
      P=MCODE(1)
      GOTO 5
    4 P=P+1
    5 OPND = MCODE(P)/100
      OP=MCODE(P)-OPND*100
      ICYCLS=ICYCLS+1
      MPNT2=MPOINT+4
      IF(LTRACE(ICYCLS)) WRITE(ISOUT,7) ICYCLS,LASTP,LOP,NMOP(LOP),LOPND
     1,S,M,MPOINT,IROW,(MCODE(I),I=MPOINT,MPNT2),LMCODE(4*(MPOINT+4)+1)
     2,(INBUF(I),I=MROW,IROW)
      LASTP=P
      LOP=OP
      LOPND=OPND
    7 FORMAT(1X,3I5,1X,A4,I5,2L5,2I6,5X,I4,4A4,L5,2X,40A1)
C
      GO TO ( 10, 20, 30, 40, 50, 60, 70, 80, 90,100,110,120,130,140,
C           SET NOB RDB TST  ID NUM  SR CLL   R  BT   B  BF  BE OUT
     1      150,160,170,180,190,200,210,220,230,240,250,260,270,280,
C            CL  CI GN  COM LID ERP  LB MS  ST  CLS HOS CS  UP  CIO
     2      290,410,410,410,410,410,410,410,370,410,410,410), OP
C           EOF FOP  OP CIP CLP POT  HV HLV REM XYZ XYZ XYZ
C
  410 WRITE(ISOUT,131) LNBR,ICYCLS,ICNT,INBUF,IPOINT
      STOP 410
C
C*****************************************************************
C (SET)   TURN MAIN SWITCH ON
C*****************************************************************
   10 S=.TRUE.
      GO TO 4
C
C*****************************************************************
C (NOB)   TURN BLANK MODE ON (BLANK IS REDUNDANT SYMBOL)
C*****************************************************************
```

```
   20 M=.TRUE.
      GOTO 4
C
C*************************************************************************
C (RDB)   TURN BLANK MODE OFF (BLANK IS SYNTACTIC UNIT DELIMITER)
C*************************************************************************
   30 M=.FALSE.
      GOTO 4
C
C*************************************************************************
C (TST)   TEST FOR A STRING IN INPUT
C*************************************************************************
   40 CALL INPUT
      I=IROW-1
      DO 43 J=1,OPND
   41 I=I+1
      IF (INBUF(I) .NE. IBL) GO TO 42
      IF (M)  GOTO 41
   42 LLITL=LMCODE(4*P+J)
      IF (INBUF(I).NE. LITL) GO TO 44
   43 CONTINUE
      IROW=I+1
      S=.TRUE.
   44 P=P+(OPND-1)/4+1
      GO TO 4
C
C*************************************************************************
C  (ID)   TEST FOR IDENTIFIER IN INPUT
C*************************************************************************
   50 CALL INPUT
      IF (.NOT. LETTER(INBUF(IROW))) GOTO 4
   51 IROW=IROW+1
      IF (LETTER(INBUF(IROW))) GOTO 51
      IF (DIGIT(INBUF(IROW))) GO TO 51
      S=.TRUE.
      GOTO 4
C
C*************************************************************************
C  (NUM)   TEST FOR NUMERIC STRING IN INPUT
C*************************************************************************
   60 CALL INPUT
      IF (.NOT.DIGIT(INBUF(IROW))) GO TO 4
   61 IROW=IROW+1
      IF (DIGIT(INBUF(IROW))) GO TO 61
      IF (INBUF(IROW) .NE. IPER) GO TO 63
   62 IROW=IROW+1
      IF (DIGIT(INBUF(IROW))) GO TO 62
   63 S=.TRUE.
      GO TO 4
C
C*************************************************************************
C  (SR)   TEST FOR QUOTED STRING IN INPUT
C*************************************************************************
   70 CALL INPUT
      IF (INBUF(IROW) .NE. IEQUAL) GO TO 4
   71 IROW=IROW+1
      IF(IROW .GE. 73) CALL INPUT
      IF (INBUF(IROW) .NE. IEQUAL) GO TO 71
```

```
      IROW=IROW+1
      IF (INBUF(IROW) .EQ. IEQUAL) GO TO 71
      S=.TRUE.
      GO TO 4
C
C*****************************************************************
C  (CLL)  RECURSIVE CALL
C*****************************************************************
   80 MPOINT=MPOINT-6
      IF (MPOINT .LE. LENGTH) GOTO 82
      MCODE(MPOINT) =P
      IF(M) MCODE(MPOINT)=-MCODE(MPOINT)
      MCODE(MPOINT+1)=IBL
      MCODE(MPOINT+2)=IBL
      MCODE(MPOINT+3)=IBL
      MCODE(MPOINT+4)=IBL
      LMCODE(4*(MPOINT+4)+1)=.FALSE.
      P=OPND
      GO TO 5
   82 WRITE(ISOUT,84)
   84 FORMAT(' M420 *** TOO MANY RECURSIONS ***')
      STOP 420
C
C*****************************************************************
C (R)   SUBROUTINE RETURN
C*****************************************************************
   90 IF (MPOINT .GE. MCLIM) GOTO 91
      P=IABS(MCODE(MPOINT))
      M=P .NE. MCODE(MPOINT)
      MPOINT=MPOINT+6
      GOTO 4
   91 IF(EFLG) RETURN
      IF(CDL) GOTO 92
      IF(NCARD) GOTO 94
      RETURN
   92 ICOUNT=1
   94 ENDFILE IPRI
      REWIND IPRI
      CALL PRINT
      RETURN
C
C*****************************************************************
C  (BT)   BRANCH IF MAIN SWITCH ON
C*****************************************************************
  100 IF (.NOT. S) GO TO 4
      P=OPND
      GO TO 5
C
C*****************************************************************
C (B)   BRANCH UNCONDITIONALLY
C*****************************************************************
  110 P=OPND
      GO TO 5
C
C*****************************************************************
C  (BF)   BRANCH IF MAIN SWITCH OFF
C*****************************************************************
  120 IF (S) GO TO 4
```

```
      P=OPND
      GO TO 5
C
C*****************************************************************
C  (BE)   STOP, DIAGNOSTIC OR BACKUP IF MAIN SWITCH IS OFF
C         - BACKUP IF BACKUP SWITCH IS ON
C         - DIAGNOSTIC IF ERROR SWITCH IS ON
C         - STOP OTHERWISE
C*****************************************************************
  130 IF(S) GOTO 4
      IF(UP) GOTO 137
      IPOINT(IROW)=ISTERN
      WRITE(ISOUT,131) LNBR,ICYCLS,ICNT,INBUF,IPOINT
  131 FORMAT(' M440 *** SYNTAX ERROR *** TRACE =',I5,', FTRACE =',I6,
     1', PROGRAM LENGTH =',I6,/,1X,80A1,/,1X,80A1)
      IPOINT(IROW)=IBL
      IF(ERR) GOTO 134
      WRITE(ISOUT,133)
  133 FORMAT(' M441 *** ERROR STOP BY STOP MODE ***')
      STOP 441
  134 IROW=73
      EFLG=.TRUE.
      ECNT=ECNT+1
      CALL INPUT
  135 IF(LMCODE(4*(MPOINT+4)+1)) GOTO 136
      MPOINT=MPOINT+6
      IF(MPOINT .LT. MCLIM) GOTO 135
      STOP 460
  136 P=IABS(MCODE(MPOINT))
      M=P .NE. MCODE(MPOINT)
      GOTO 5
  137 IROW=INUP
      MROW=MINUP
      UP=.FALSE.
      INLN=INLN+1
      WRITE(ISOUT,1301) UP,INUP,MINUP,INLN,OUTUP,OUTLN
 1301 FORMAT(L3,' INUP',I3,' MINUP',I3,' INLN',I3,' OUTUP',I3,' OUTLN',I
     13)
      DO 138 I=1,INLN
  138 BACKSPACE IFILE
      READ(IFILE,254) INBUF
      WRITE(ISOUT,252) INBUF
      ICNT=OUTUP
      IF(.NOT. LIPRI(LNBR)) GOTO 4
      IF(OUTLN .EQ. 0) GOTO 4
      DO 139 I=1,OUTLN
  139 BACKSPACE IPRI
      LNBR=LNBR-OUTLN
      GOTO 4
C
C*****************************************************************
C  (OUT)   PRINT LINE IN OUTPUT BUFFER
C*****************************************************************
  140 LNBR=LNBR+1
      IF(LISOUT(LNBR)) WRITE(ISOUT,141) LNBR,ICYCLS,(OUT(I),I=1,30),
     1              (INBUF(I),I=MROW,IROW)
  141 FORMAT(5X,'LN',I4,1X,'(',I6,')',1X,30A1,1X,30A1)
      IF(LIPRI(LNBR) .AND. UP) OUTLN=OUTLN+1
```

```
      IF(LIPRI(LNBR)) WRITE(IPRI,142) (OUT(I),I=1,72),ISTR,LNBR
  142 FORMAT(72A1,'SIM',A1,I4)
      DO 143 L=1,4
  143 LAS(4*(ICNT-1)+L)=LCUT(4*(L-1)+1)
      IF (IASS(ICNT) .NE. IBL) GO TO 148
      ICNT =ICNT+1
      DO 144 L=7,10
  144 LAS(4*(ICNT-1)+L-6)=LOUT(4*(L-1)+1)
      ICNT=ICNT+1
      DO 145 L=11,72
  145 LAS(4*(ICNT-1)+L-10)=LOUT(4*(L-1)+1)
      DO 146 I=1,16
      IF (IASS(ICNT) .EQ. IBL) GO TO 147
  146 ICNT=ICNT+1
  147 IASS(ICNT) =0
  148 ICNT=ICNT+1
      DO 149 I=1,72
  149 OUT(I)=IBL
      OROW=7
      IF(ICNT .GE. MAXOUT) GO TO 1040
      GO TO 4
 1040 WRITE(ISOUT,1041) INBUF
 1041 FORMAT(' M450 *** DIMENSION OF IASS -(9500)- TOO SMALL',/,1X,
     180A1)
      STOP 450
C
C*****************************************************************
C  (CL)  COPY LITERAL INTO OUTPUT BUFFER
C*****************************************************************
  150 DO 151 I=1,OPND
      LLITL=LMCODE(4*P+I)
      OUT(OROW) = LITL
  151 OROW=OROW+1
      OROW=OROW+1
      P=P+(OPND-1)/4+1
      GO TO 4
C
C*****************************************************************
C  (CI)   COPY INPUT INTO OUTPUT BUFFER
C*****************************************************************
  160 I=IROW-1
      DO 161   J=MROW,I
      OUT(OROW)=INBUF(J)
  161 OROW=OROW+1
      OROW=OROW+1
      GOTO 4
C
C*****************************************************************
C  (GN)   GENERATE LABEL IN BOX 1,2,3 OR 4
C  IF NOT ALREADY THERE AND PRINT IT INTO OUTPUT BUFFER
C*****************************************************************
  170 J=MPOINT+CPND
      IF (MCODE(J) .NE. IBL) GO TO 174
      LONE=LONE+1
      IF(LONE .LT. 10) GOTO 172
      LONE=0
      LTEN=LTEN+LMASK
      IF (LTEN.LT.10*LMASK) GO TO 172
```

```
      LTEN=0
      LHUN=LHUN+LMASK**2
      IF(LHUN .LT. 10*LMASK**2) GO TO 172
      LHUN=0
      LALPHA=LALPHA+LMASK**3
  172 MCODE(J)=LONE+LTEN+LHUN+LALPHA
  174 DO 176 I=1,4
  176 LLBL(4*(I-1)+1)=LMCODE(4*(J-1)+I)
      DO 178 J=1,4
  178 OUT(OROW+J-1)=LBL(J)
      OROW=OROW+6
      GOTO 4
C
C*************************************************************************
C  (COM)    TEST FOR A COMMENT STRING (NO FURTHER OPERATION)
C*************************************************************************
  180 CALL INPUT
      I=IROW-1
      DO 183 J=1,OPND
  181 I=I+1
      IF(INBUF(I) .NE. IBL) GOTO 182
      IF(M) GOTO 181
  182 LLITL=LMCODE(4*P+J)
      IF(INBUF(I) .NE. LITL) GOTO 188
  183 CONTINUE
  189 I=I+1
      DO 187 IROW=I,72
      I=IROW-1
      DO 186 J=1,OPND
  184 I=I+1
      IF(INBUF(I) .NE. IBL) GOTO 185
      IF(M) GOTO 184
  185 LLITL=LMCODE(4*P+J)
      IF(INBUF(I) .NE. LITL) GOTO 187
  186 CONTINUE
      IROW=I+1
      S=.TRUE.
      GOTO 188
  187 CONTINUE
      I=73
      CALL INPUT
      GOTO 189
  188 P=P+(OPND-1)/4 + 1
      GOTO 4
C
C*************************************************************************
C   (LID)    TEST FOR LABEL IDENTIFIER IN INPUT
C*************************************************************************
  190 CALL INPUT
      IF(.NOT. LETTER(INBUF(IROW))) GOTO 195
      I=IROW
  191 I=I+1
      IF(LETTER(INBUF(I))) GOTO 191
      IF(DIGIT(INBUF(I))) GOTO 191
      I=I-1
      DO 194 J=1,OPND
  192 I=I+1
      IF(INBUF(I) .NE. IBL) GOTO 193
```

```
      IF(M) GOTO 192
  193 LLITL=LMCODE( 4*P+J)
      IF(INBUF(I) .NE. LITL) GOTO 195
      INBUF(I)=IBL
  194 CONTINUE
      IROW=I+1
      S=.TRUE.
  195 P=P+(OPND-1)/4 + 1
      GOTO 4
C
C*************************************************************************
C  (ERP)    ERROR RETURN POINT
C*************************************************************************
  200 LMCODE(4*(MPOINT+4)+1)=.TRUE.
      ERR=.TRUE.
      S=.TRUE.
      GOTO 4
C
C*************************************************************************
C  (LB)   PREPARE TO ISSUE LABEL
C*************************************************************************
  210 OROW=1
      GOTO 4
C
C*************************************************************************
C  (MS)    COPY LITERAL INTO MESSAGE FILE (STANDARD OUTPUT)
C*************************************************************************
  220 L=0
      OPND=(OPND-1)/4+1
      DO 222 I=1,OPND
      L=L+1
  222 MS(L)=MCODE(P+I)
      IPOINT(IROW)=ISTERN
      WRITE(ISOUT,224) INBUF,IPOINT,(MS(I),I=1,L)
  224 FORMAT(' M430 *** SYNTAX ERROR ***',/,1X,80A1,/,1X,80A1,/,1X,20A4)
      IPOINT(IROW)=IBL
      IROW=73
      EFLG=.TRUE.
      ECNT=ECNT+1
      S=.TRUE.
      P=P+OPND
      GOTO 4
C
C*************************************************************************
C   (ST)  COPY INPUT INTO COMMON REGISTER 1,2 OR 3
C*************************************************************************
  230 K=OPND
      I=IROW-1
      MARS=0
      DO 231 J=MROW,I
      MARS=MARS+1
  231 STREG(K,MARS)=INBUF(J)
      MARSK(K)=MARS
      GOTO 4
C
C*************************************************************************
C    (CLS)  COPY LITERAL INTO COMMON REGISTER 1
C*************************************************************************
```

```
  240 MARS=0
      DO 241 I=1,OPND
      J=4*P+I
      LLITL=LMCODE(J)
      MARS=MARS+1
  241 STREG(1,MARS)=LITL
      MARSK(1)=MARS
      GOTO 4
C
C*****************************************************************
C    (HOS)   HOST RECOGNIZER
C*****************************************************************
  250 K=(MCODE(P+1)-CHMASK)/LMASK**3
      J=MCODE(P+2)
  251 CALL INPUT
      IF(INBUF(1) .EQ. -1) GOTO 256
      IF(INBUF(K) .NE. IBL .AND. INBUF(1) .NE. J) GOTO 256
      IF(LISOUT(LNBR)) WRITE(ISOUT,252) (INBUF(I),I=1,80)
  252 FORMAT(1X,80A1)
      IF(INBUF(76) .EQ. ISTR) INBUF(76)=IBL
      IF(LIPRI(LNBR)) WRITE(IPRI,254) (INBUF(I),I=1,80)
  254 FORMAT(80A1)
      IROW=80
      S=.TRUE.
      GOTO 251
  256 P=P+(OPND-1)/4+1
      GOTO 4
C
C*****************************************************************
C    (CS)   COPY CONTENT OF COMMON REGISTER 1,2 OR 3
C      INTO OUTPUT BUFFER
C*****************************************************************
  260 K=OPND
      MARS=MARSK(K)
      DO 261 J=1,MARS
      OUT(OROW)=STREG(K,J)
  261 OROW=OROW+1
      OROW=OROW+1
      GOTO 4
C
C*****************************************************************
C    (UP)   TURN BACKUP SWITCH ON
C*****************************************************************
  270 UP=.TRUE.
      INUP=IROW
      MINUP=MROW
      INLN=0
      OUTUP=ICNT
      OUTLN=0
      GOTO 4
C
C*****************************************************************
C    (CIO)  COPY INPUT LINE INTO OUTPUT BUFFER STARTING FROM MROW
C*****************************************************************
  280 I=OROW
      DO 282 J=MROW,72
      OUT(I)=INBUF(J)
      IF(I .GE.72) GOTO 284
```

```
  282 I=I+1
  284 GOTO 4
C
C*****************************************************************************
C     (EOF)  TEST IF END OF FILE IN INPUT OCCURED
C*****************************************************************************
C
  290 IF(INBUF(1) .NE. -1) GOTO 4
      S=.TRUE.
      GOTO 4
C
C*****************************************************************************
C        (REM) COPY REMARK ON OUTPUT
C*****************************************************************************
  370 IREM=LONE+LTEN+LHUN+REM
      DO 371 J=1,4
      LLBL(4*(J-1)+1)=LIREM(J)
      OUT(OROW)=LBL(J)
      OROW=OROW+1
  371 CONTINUE
      GOTO 4
C
      END
```

W. Kneis, META-II/X-System

```
      SUBROUTINE INPUT
C     IROW IS POSITION MARKER OF FIRST CHARACTER .NE. BLANK
C     FIND FIRST CHARACTER .NE. BLANK
      IMPLICIT INTEGER (A-Z)
      COMMON /LIMITS/ CIL,COL,COL1,COL2,CDL,CDL1,CDL2,ICYCLS,
     1                CTL,CTL1,CTL2,ALL,ALL1,ALL2,MSAVE,NSAVE,NCARD,
     2                IND,JCNTR,BRA,COMMA,KET,CHMASK,
     3                NA,NI,NJ,NR,NS,NZ,NULL,NINE,
     4                KOPS,IADR,NMOP(40),IEND,IEL,IEQUAL,ISTERN,IPER,
     5                ISTR,LOPF,LADRF
      COMMON /META/IP,MCODE(4000),LBL(4),INBUF(80),IROW,S,MROW,JBUF(80),
     1  OROW,ICCD,IFILE,ISIN,ISOUT,IPRI,IPUN,ICAR,IPAGE,ICOUNT,LNBR,UP
     2  ,INLN,OUTUP,ERR,EFLG,ECNT
      LOGICAL S,UP
      S=.FALSE.
      IROW=IROW-2
    1 IROW=IROW+1
      IF (IROW.GE.72) GO TO 3
      IF (INBUF(IROW+1) .EQ. IBL) GO TO 1
    2 IROW=IROW+1
      MROW=IROW
      RETURN
    3 IROW=0
      READ(IFILE,10,END=5) (INBUF(I),I=1,80)
      IF(UP) INLN=INLN+1
   10 FORMAT(80A1)
      IF (INBUF(1)-IBL) 2,1,2
    5 INBUF(1)=-1
      GOTO 2
      END
```

```
      SUBROUTINE PRINT
C     ACTUALLY ONLY PRINTING OF OUTPUT LISTING ALLOWED
C     CONTROL TO THIS ROUTINE ONLY WILL BE GIVEN, WHEN NO ERROR
C     IN COMPILING HAS OCCURED.
      IMPLICIT INTEGER (A-Z)
      REAL*8 DDATE,DTIME
      COMMON /LIMITS/ CIL,COL,COL1,COL2,CDL,CDL1,CDL2,ICYCLS,
     1                CTL,CTL1,CTL2,ALL,ALL1,ALL2,MSAVE,NSAVE,NCARD,
     2                IND,JCNTR,BRA,COMMA,KET,CHMASK,
     3                NA,NI,NJ,NR,NS,NZ,NULL,NINE,
     4                KOPS,IADR,NMOP(40),IEND,IBL,IEQUAL,ISTERN,IPER,
     5               ISTR,LOPF,LADRF
      COMMON /ASSEM/ DDATE,DTIME,IASS(9500),ICNT
      COMMON /META/IP,MCODE(4000),LBL(4),INBUF(80),IROW,S,MROW,JBUF(80),
     1  OROW,ICCD,IFILE,ISIN,ISOUT,IPRI,IPUN,ICAR,IPAGE,ICOUNT,LNBR,UP
     2  ,INLN,OUTUP,ERR,EFLG,ECNT
      LOGICAL CIL,CDL,COL,CTL,ALL,MSAVE,NSAVE,NCARD
      JLOPF=LOPF-1
  100 READ (IPRI,110,END=300) (INBUF(I),I=1,80)
  110 FORMAT(80A1)
      IF(INBUF(1) .NE. ISTERN .AND. INBUF(76) .EQ. ISTR) GOTO 120
      DO 130 J=1,72
  130 JBUF(J)=INBUF(J)
      GOTO 250
  120 DO 200 J=1,72
  200 JBUF(J)=IBL
      DO 210 J=1,JLOPF
      JBUF(J)=INBUF(J)
      IF (JBUF(J) .EQ. IBL) GOTO 212
  210 CONTINUE
  212 JOT=J+1
      DO 213 J=JOT,LOPF
      IF (INBUF(J) .NE. IBL) GOTO 215
  213 CONTINUE
  215 JOT=J
      JJ=LOPF-1
      DO 214 J=JOT,LADRF
      JJ=JJ+1
      JBUF(JJ)=INBUF(J)
      IF (JBUF(JJ) .EQ. IBL) GOTO 216
  214 CONTINUE
  216 JOT=J+1
      JJ=LADRF-1
      DO 218 J=JOT,72
      JJ=JJ+1
      JBUF(JJ)=INBUF(J)
      IF (JBUF(JJ) .EQ.  IBL) JJ=JJ-1
  218 CONTINUE
  250 IF(ICOUNT .GT. 1) GOTO 220
      WRITE(ISOUT,219) DDATE,IPAGE
      IPAGE=IPAGE+1
  219 FORMAT(1H1,/,' META II/X - SIMULATOR - OUTPUT',T50,'DATE ',A8,
     1       T78,'PAGE ',I3,/)
  220 IF(ICOUNT .EQ. 55) ICOUNT=0
      IF(.NOT. CIL) GOTO 230
      WRITE(ISOUT,240) (JBUF(I),I=1,72),(INBUF(I),I=73,80)
      ICOUNT=ICOUNT+1
  240 FORMAT(5X,80A1)
```

```
230 IF(NCARD) WRITE(ICCD,110) (JBUF(I),I=1,72),(INBUF(I),I=73,80)
    GOTO 100
300 REWIND IPRI
    RETURN
    END
```

W. Kneis, META-II/X-System

```
      SUBROUTINE ASEMBL
C     ASSEMBLING OF NMEMONIC ASSEMBLER CODE INTO SIMULATOR
C     MACHINE LANGUAGE
C     ESTABLISHING OF SYMBOLS IN THE SYMBOL TABLE (ONLY 4 CHARACTERS)
C     LOADING THE MACHINE PROGRAM INTO MCODE=KODORG=KODE IN THE
C     BLOCK DATA AREA
      IMPLICIT INTEGER (A-Z)
      REAL*8 DDATE,DTIME
      COMMON /LIMITS/ CIL,CCL,COL1,COL2,CDL,CDL1,CDL2,ICYCLS,
     1                CTL,CTL1,CTL2,ALL,ALL1,ALL2,MSAVE,NSAVE,NCARD,
     2                IND,JCNTR,BRA,COMMA,KET,CHMASK,
     3                NA,NI,NJ,NR,NS,NZ,NULL,NINE,
     4                KOPS,IADR,NMOP(40),IEND,IBL,IEQUAL,ISTERN,IPER,
     5               ISTR,LCPF,LADRF
      COMMON /ASSEM/ DDATE,DTIME,IASS(9500),ICNT
      COMMON /ORIGIN/ MAXOUT,MCLIM,LEN,LMASK,MCODE(4000)
      COMMON /META/P,LBL(2002),LADR(2002),INBUF(80),IROW,S,MROW,JBUF(80)
     1 ,OROW,ICCD,IFILE,ISIN,ISOUT,IPRI,IPUN,ICAR,IPAGE,ICOUNT,LNBR,UP
     2  ,INLN,CUTUP,ERR,EFLG,ECNT
      LOGICAL EFLG,CIL,CDL,CCL,CTL,ALL,MSAVE,NSAVE,NCARD
      LOGICAL*1 LAS(1),LINBUF(1),LJBUF(1)
      LOGICAL LIST
      EQUIVALENCE (IASS(1),LAS(1)),(INBUF(1),LINBUF(1)),
     1            (JBUF(1),LJBUF(1))
      LIST(IP)=ALL .AND. IP .GE. ALL1 .AND. IP .LE. ALL2
C*****************************************************************
C     FIRST PASS - LABEL IDENTIFICATION AND PROGRAM LENGTH
C                  STRING EXPRESSIONS ARE HANDLED IN MULTIPLES
C                  OF FOUR BYTES; REST IS FILLED WITH BLANKS
C                  (LEFT JUSTIFIED)
C                  WARNING:
C                  ********
C                  THE LENGTH OF INSTRUCTIONS USING STRING EXPRESSIONS
C                  AS OPERANDS MUST HAVE BEEN DECLARED IN THIS STEP
C*****************************************************************
      MLBLS=(2002)
      MIP=(4000)
      EFLG=.FALSE.
      ECNT=0
      NLBLS=0
      IP=1
      IC=0
      ICOUNT=0
    1 IC=IC+1
      IF (IASS(IC) .EQ. IBL) GO TO 2
      NLBLS=NLBLS+1
      IF(NLBLS .LE. MLBLS) GOTO 5
      WRITE(ISOUT,14)
   14 FORMAT(' M810 *** NUMBER OF LABELS - (2002) - EXCEEDED ***')
      STOP 810
    5 LBL(NLBLS)=IASS(IC)
      LADR(NLBLS)=IP
      GOTO 1
    2 IP=IP+1
      IF(IP .LE. MIP) GOTO 15
      WRITE(ISOUT,16)
   16 FORMAT(' M850 *** MACHINE BUFFER, MCODE(4000), TOO SMALL ***')
      STOP 850
```

```
   15 IC=IC+1
      IF (IASS(IC) .EQ. IEND) GO TC 20
      IF( IASS( IC).EQ.NMOP(  4) .OR. IASS( IC).EQ.NMOP(15) .OR.
     1    IASS( IC).EQ.NMOP(18) .OR. IASS( IC).EQ.NMOP(19) .OR.
     2    IASS(IC).EQ.NMOP(22) .OR. IASS( IC).EQ.NMOP( 25) ) GOTO 4
    3 IC=IC+1
      IF (IASS(IC)) 3,1,3
C
    4 IC=IC+1
      DO 101 I=1,64
  101 INBUF(I)=IBL
      DO 100 I=1,64
  100 LINBUF( 4*( I-1)+1)=LAS( 4*( IC-1)+I)
      DO 6 K=1,64
      IF (INBUF(K) .EQ. IEQUAL) GOTO 7
    6 CONTINUE
    7 J=K+1
    8 IF (J .GE. 64) GO TO 10
      IF (INBUF(J) .NE. IEQUAL) GOTO 9
      IF (INBUF(J+1) .NE. IEQUAL) GOTO 10
      J=J+1
      K=K+1
    9 J=J+1
      GOTO 8
   10 IP=IP+(J-K+2)/4
      GO TO 3
C*****************************************************************
C     SECOND PASS - REPLACING MNEMONIC OPERATION CODES BY NUMERIC VALUES
C                   ADDRESS SYMBOL IDENTIFICATION IN OPERATION FIELD
C                   AND EXPLICIT STRING HANDLING
C                   (TRANSFER IN OBJECT CODE AREA)
C*****************************************************************
   20 LEN=IP-1
      IC=0
      IP=1
      LABEL=IBL
   21 ICOUNT=0
      ICOUNT=ICOUNT+1
      IF(.NOT. LIST(IP) .OR. ICOUNT .GT. 1) GOTO 23
      WRITE(ISOUT,22)DDATE,IPAGE
      IPAGE=IPAGE+1
   22 FORMAT(1H1,/,' META II/X - ASSEMBLER - OUTPUT',T50,'DATE ',A8,
     1  T78,'PAGE ',I3,/)
   23 IC=IC+1
      IF( IASS( IC) .EQ. IBL) GO TO 25
C
      IF(LABEL .NE. IBL .AND. LIST( IP)) WRITE(ISOUT,24) LABEL
   24 FORMAT(34X,A4)
      LABEL=IASS( IC)
      GO TO 23
C
   25 IC=IC+1
      IOP=IASS(IC)
      IF (IOP .EQ. IEND) RETURN
      K=0
      IF (IOP .EQ. IADR) GO TO 40
      DO 26 K=1,KOPS
      IF (IOP .EQ. NMOP(K)) GO TO 28
```

```
   26 CONTINUE
      K=0
   28 IF(ICCUNT .GE. 55) ICCUNT=0
      ICOUNT=ICOUNT+1
      IF( .NOT. LIST(IP) .OR. ICOUNT .GT. 1 ) GOTO 29
      WRITE( ISOUT,22) DDATE,IPAGE
      IPAGE=IPAGE+1
   29 GO TO ( 30, 30, 30, 50, 30, 30, 30, 40, 30, 40, 40, 40, 30, 30,
C               1   2   3   4   5   6   7   8   9  10  11  12  13  14
C             SET NCB RDB TST  ID NUM  SR CLL   R  BT   B  BF  BE OUT
     1         50, 30, 70, 50, 50, 30, 30, 50, 70, 50, 50, 70, 30, 30,
C              15  16  17  18  19  20  21  22  23  24  25  26  27  28
C              CL  CI  GN COM LID ERP  LB  MS  ST CLS HCS  CS  UP CIC
     2         30,170,170,170,170,170,170,170, 30,170,170,170), K
C              29  30  31  32  33  34  35  36  37  38  39  40
C             ECF FCP  OP CIP CLP PCT  HV HLV REM XYZ XYZ XYZ
C
      EFLG=.TRUE.
      ECNT=ECNT+1
      WRITE(ISOUT,27) IOP
   27 FORMAT(' M820 *** INVALID OP CODE (',A4,') ***')
  271 IP=IP+1
      IC=IC+1
      IF(IASS(IC))271,23,271
C*****************************************************************
C     NO OPERAND
C*****************************************************************
   30 MCODE(IP)=K
      IF (LIST(IP)) WRITE( ISOUT,31) IP,K,LABEL,IOP
   31 FORMAT(5X,'LN',I5,I5,16X,2A4)
      LABEL=IBL
      IC=IC+1
      IP=IP+1
      IF ( IOP .EQ. NMOP(9)) GO TO 21
      GO TO 23
C*****************************************************************
C     OPERAND IS AN ADDRESS SYMBOL
C*****************************************************************
   40 IC=IC+1
      ISYMA=IASS( IC)
      DO 41 M=1,NLBLS
      IF (LBL(M) .EQ. ISYMA) GO TO 43
   41 CONTINUE
      WRITE(ISOUT,42) ISYMA
   42 FORMAT(' M830 *** UNDEFINED SYMBOL (',A4,') ***')
      EFLG=.TRUE.
      ECNT=ECNT+1
   43 MCODE( IP)=LADR(M)*100+K
      IF (K .EQ. 0) MCODE(IP) =LADR(M)
      IF (LIST( IP)) WRITE( ISOUT,44) IP,K,LADR(M),LABEL,IOP,ISYMA
   44 FORMAT(5X,'LN',I6,I5,I4,12X,3A4)
      LABEL=IBL
      IP=IP+1
      IC=IC+1
      GOTO 23
C*****************************************************************
C     OPERAND IS A STRING EXPRESSION
C*****************************************************************
```

```
   50 IC=IC+1
      DO 110 I=1,64
  110 LINBUF(4*(I-1)+1)=LAS(4*(IC-1)+I)
      N=0
      DO 51 JJ=1,64
      IF (INBUF(JJ) .EQ. IEQUAL) GOTO 52
   51 CONTINUE
   52 J=JJ+1
   53 IF (J .GE. 64) GO TO 55
      IF (INBUF(J) .NE. IEQUAL) GOTO 54
      IF (INBUF(J+1) .NE. IEQUAL) GOTO 55
      J=J+1
   54 N=N+1
      JBUF(N)=INBUF(J)
      J=J+1
      GO TO 53
   55 MCODE(IP)=N*100+K
      IF (LIST(IP)) WRITE(ISOUT,56) IP,K,N,LABEL,IOP,(INBUF(L),L=JJ,J)
   56 FORMAT(5X,'LN',I6,I5,I4,12X,2A4,64A1)
      LABEL=IBL
      DO 120 L=1,N
  120 LINBUF(L)=LJBUF(4*(L-1)+1)
      N=(N+3)/4
      DO 59 I=1,N
      IP=IP+1
      ICOUNT=ICOUNT+1
      IF (LIST(IP)) WRITE(ISOUT,58) IP,INBUF(I)
   58 FORMAT(5X,'LN',I6,9X,' ',A4,' ')
   59 MCODE(IP)=INBUF(I)
      DO 61 I=1,64
      INBUF(I)=IBL
   61 JBUF(I)=IBL
      IP=IP+1
   60 IC=IC+1
      IF (IASS(IC)) 60,23,60
C*****************************************************************
C     OPERAND IS A LITERAL
C*****************************************************************
   70 IC=IC+1
      N=(IASS(IC)-CHMASK)/LMASK**3
      MCODE(IP)=N*100+K
      IF(LIST(IP)) WRITE(ISOUT,72) IP,K,N,LABEL,IOP,IASS(IC)
   72 FORMAT(5X,'LN',I6,I5,I4,12X,3A4)
      LABEL=IBL
      IP=IP+1
      IC=IC+1
      GOTO 23
C*****************************************************************
C     UNALLOWED OPERATIONCODES
C*****************************************************************
  170 WRITE(ISOUT,171) IOP
  171 FORMAT(' M840 *** UNALLOWED OP CODE (',A4,') ***')
      EFLG=.TRUE.
      ECNT=ECNT+1
      IP=IP+1
      IC=IC+1
      GOTO 23
      END
```

A-8 Jobcontrol Beispiel fuer IBM/370

```
//ZYK318MT JOB (0318,060,P0000),KNEIS,REGION=200K
//PRINT EXEC FGCG,COMP=X
//C.SYSPRINT DD DUMMY
//C.SYSIN DD DISP=SHR,DSN=TS0318.META9.FORT
//G.FT05F001 DD DISP=SHR,
// DSN=TS0318.MDECK.DATA(EX70),LABEL=(,,,IN)
//G.FT06F001 DD UNIT=(CTC,,DEFER)
//G.FT04F001 DD UNIT=SYSDA,DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680),
// SPACE=(1680,(50,10))
//G.FT07F001 DD UNIT=SYSDA,DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680),
// SPACE=(1680,(50,10))
//G.FT08F001 DD DISP=SHR,DSN=TS0318.IPUN.DATA
//G.FT09F001 DD DISP=SHR,
// DSN=TS0318.BDECK.DATA(EX40),LABEL=(,,,IN)
//G.FT02F001 DD UNIT=SYSDA,DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680),
//  SPACE=(1680,(50,10))
//
```